AFRL-SR-AR-TR-02-0164

A Process Engineering Approach to the Development and Integration of Intrusion Detection Techniques

Nong Ye
Arizona State University

20020517 023

Air Force Research Laboratory
Air Force Office of Scientific Research
Arlington, Virginia

REPORT DOCUMENTATION PAGE

AFRL-SR-AR-TR-02-

0164

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE March 1, 2002	3. REPORT TYPE AND DATES COVERED Final Technical Report 1 October 1998 – 31 December 2001

4. TITLE AND SUBTITLE A Process Engineering Approach to the Development and Integration of Intrusion Detection Techniques	5. FUNDING NUMBERS G F49620-99-1-0014
6. AUTHORS Nong Ye	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Arizona State University Box 873503 Tempe, AZ 85287-3503	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFOSR/NM 801 N. Randolph St Arlington, VA 22203-1977	10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The objectives of this project are to: 1)establish an integration infrastructure for intrusion detection, based on a process engineering approach; 2) investigate system-level intrusion detection techniques for the fusion and correlation of local information about intrusions, based on the integration infrastructure for intrusion detection; and 3) develop an integrated intrusion detection system as a concept technological demonstration of the integration infrastructure and system-level intrusion detection techniques.
In this final project report, we present the architecture of an integrated intrusion detection system based on the process engineering approach, as well as various intrusion detection techniques that are employed in this integrated intrusion detection system. The testing results of these intrusion detection techniques are also illustrated to demonstrate their intrusion detection performance. Various attributes of system activity data for intrusion detection are tested and compared.

14. SUBJECT TERMS Intrusion detection, statistical process control, data mining, information security, computer audit data	15. NUMBER OF PAGES 281
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

A Process Engineering Approach to the Development and Integration of Intrusion Detection Techniques

GRANT NO.:	**F49620-99-1-0014**
PI:	**Nong Ye**
SPONSOR:	**AFOSR**
REPORTING PERIOD:	**1 October 1998 - 31 March 2001**

Table of Contents

List of Figure Captions

List of Table Captions

Abstract

The objectives of the proposed work are to: 1) establish an integration infrastructure for intrusion detection, based on a process engineering approach; 2) investigate system-level intrusion detection techniques for the fusion and correlation of local information about intrusions, based on the integration infrastructure for intrusion detection; and 3) develop an integrated intrusion detection system as a concept technological demonstration of the proposed integration infrastructure and system-level intrusion detection techniques. In this final project report, we present the architecture of an integrated intrusion detection system based on the process engineering approach, as well as various intrusion detection techniques that are employed in this integrated intrusion detection system. The testing results of these intrusion detection techniques are also illustrated to demonstrate their intrusion detection performance. Various attributes of system activity data for intrusion detection are tested and compared.

Objectives

An intrusion is a set of actions that intend to compromise the security (i.e., integrity, confidentiality, availability, accountability, functionality, and survivability) of a computer network system. Intrusions to the secure operation of a computer network system may take many forms: external attacks, internal misuses, network-based attacks, host-based attacks, information gathering, access to privileged services, denial of service, and so on. The nation's defense systems are increasingly dependent on computer network systems for information communication and processing. Connected information systems are vulnerable to intrusions. Threats to computer network systems would undermine the nation's technology-based strategy of military superiority. Therefore, the importance of computer network security has been well acknowledged.

Existing techniques for network security management generally fall in two categories: prevention and detection. Intrusion prevention plans and enforces security policies that protect a computer network system against intrusions at potential points of break-in. Security policies specify authorized and/or unauthorized network activities concerning many aspects of network security, such as access control, confidentiality, data integrity, non-repudiation, accountability, and so on. Requests for unauthorized network activities are rejected. A number of techniques are available to support security policies, including encryption, authentication, firewall, application gateways, and other well-known network management protocols such as DNS, SMTP and SNMP.

However, not all security needs can be specified as security policies in advance due to the unlimited, unpredictable, and uncertain nature of intrusions. For example, the distinction between unauthorized and authorized network activities can be extremely blurred. It is often difficult to determine whether a request for a network service is malicious or not. A request for a network service may be a part of a malicious attack or a normal network activity. The rejection of a request of such a nature implies a degradation of network service to the user who makes the request without a malicious intention. Some authorized network activities may be used to launch a malicious attack. Hence, a tradeoff between network security and network service must be considered when making security policies. Typically, intrusion prevention is used to provide a baseline defense against intrusions.

Intrusion detection techniques are necessary to complement intrusion prevention for discovering intrusions while they are taking place. Intrusion detection techniques deal with the monitoring, assessment, diagnosis and response of intrusions. Network activities are monitored to detect the presence of an intrusion by collecting, storing, and analyzing security-relevant data (i.e., audit data and network traffic data) which may demonstrate symptoms of intrusions. A diagnosis is performed to trace the origin, path and type of the intrusion and to assess the impact (extent and severity) of the intrusion accordingly. Diagnostic results assist in choosing responses (e.g., warning system administrators, reporting incidents, launching countermeasures to prevent further damage).

Intrusion detection techniques can be classified into five categories: statistical-based, signature-based, specification-based, prediction-based, and probability-based. Statistical-based techniques build statistical profiles (e.g., frequency distributions) for quantitative measures of the normal network behavior, and detect significant deviations of an observed network behavior from those statistical profiles. Signature-based techniques recognize well-known intrusion signatures of qualitative nature, such as strings and event sequences. Specification-based techniques represent the normal behavior (e.g., authorized operations) of privileged programs in a symbolic form (e.g., predicate logic), and check for deviations from the specification of the normal behavior. Prediction-based techniques induce the time-series relationship of network events, and compare an observed event with the anticipated event for anomaly detection. Probability-based techniques use Bayesian evidential reasoning to determine the probability of an intrusion based on an evidence for its occurrence.

Therefore, those intrusion detection techniques respond to different aspects of network activities and different kinds of intrusive behavior. For example, signature-based techniques react to only specific types of intrusions with known signatures. Prediction-based techniques concern only the temporal aspect of intrusive behavior. Not only are those techniques limited in the type of intrusive behavior, but they also are restrained in the scope of intrusive behavior. For example, probability-based techniques currently consider isolated evidences and intrusions. Existing work on both specification-based techniques and statistical-based techniques focuses on the behavioral specification and profiling for individual components (e.g., users, application programs, routers, etc.) in a computer network system. The specification and profiling can hardly scale up to the sophisticated behavior at the system level which involves multiple components and their interactions in unlimited manners. Therefore, different intrusion detection techniques offer diverse but point solutions to intrusion detection, and currently lack the power to address the system-level, structural aspect of intrusive behavior. Apparently, intrusions are not limited to known types or local scopes of network activities. When point solutions are applied to the system-level intrusive behavior, they act like diverse but potentially unreliable local sensors. System-level solutions to intrusion detection are required to correlate and fuse the outcomes of local sensors into a more reliable, global awareness of intrusions. A coherent infrastructure is also necessary to integrate system-level techniques with the existing techniques to form an integrated intrusion detection system.

The objectives of the proposed work are to:

1) Establish an integration infrastructure for intrusion detection, based on a process engineering approach;
2) Investigate system-level intrusion detection techniques for the fusion and correlation of local information about intrusions, based on the integration infrastructure for intrusion detection; and
3) Develop an integrated intrusion detection system as a concept technological demonstration (CTD) of the proposed integration infrastructure and system-level intrusion detection techniques.

In this final project report, we present the architecture of an integrated intrusion detection system based on the process engineering approach, as well as various intrusion detection techniques that are employed in this integrated intrusion detection system. The testing results of these intrusion detection techniques are also illustrated to demonstrate their intrusion detection performance.

Chapter 1: A System Architecture for Intrusion Detection

This chapter describes the architecture of an integrated intrusion detection system based on a process engineering approach. Intrusion detection techniques employed in this system are presented. The testing performance of these intrusion detection techniques is illustrated in the following chapters.

An intrusion into an information system tries to compromise the security of the system. Intrusion Detection Systems (IDSs) attempt to detect these intrusions. Specifically, this chapter discusses what an IDS requires from the target information system and how the IDS detects intrusions into the target information system. We describe the architecture of a distributed host-based IDS developed at the Information and Systems Assurance Laboratory, Arizona State University. At each host machine in the information system we install an event data collector that collects and filters data of events from the host machine. The Centralized IDS Server receives the processed data and sends them to Individual Technique Servers. These Individual Technique Servers use different intrusion detection algorithms covering both anomaly detection techniques and signature recognition techniques. Each Individual Technique Server determines an intrusion warning (IW) level for each event. The Centralized IDS Server then integrates the IW levels from the Individual Technique Servers into a composite IW level, and provides it to the security administrator.

I. INTRODUCTION

Information security has now become a critical issue with rapid development of e-commerce transaction systems over the Internet. An information system for an organization typically consists of a network of host machines. An intrusion into the information system stems either from inside the network or outside the network, and can steal classified information or create havoc in the system and halt normal user activities, incurring huge monetary and credibility losses. An Intrusion Detection System (IDS) aims in detecting intrusive activities, and gives warnings to the System Security Administrator (SSA). Existing IDSs can be categorized into three types: host-based IDS, network-based IDS and router-based IDS (Durst et al., 1999).

Host-based IDSs are usually deployed on individual host-machines to monitor activities on the host machines. Since the host-based IDSs are deployed in individual host machines, their implementation is machine-dependent, and takes away much of the computing power from the users of the host machines. If the hosts become victims from intrusions, then IDSs are brought down along with the host machines. Moreover, separate IDSs on individual host machines do not provide a direct, complete picture of distributed intrusions against a network, making it difficult for the SSA to take corrective actions. The main advantage of the host-based IDSs is that it can detect intrusions targeting the host machines from both insiders and outsiders.

Network-based IDSs are installed in some strategic computers in the network to monitor data packages sent between host machines. Network-based IDSs can detect violations of network security policy, but may not be able to process information such as not only the header portion but also the data portion of data packages so as to reveal specific intrusive activities for accurate detection. Moreover, large volumes of network traffic data present difficulty in efficiently processing such data.

Router-based IDSs are installed on routers to monitor data packages passing through routers, thus trying to prevent intrusive data packages from entering the network inside the router. Router-based IDSs are similar to network-based IDSs, and thereby suffer from similar problems.

At the Information and Systems Assurance Laboratory (ISA), Arizona State University, we have developed a distributed, host-based IDS to overcome the problems with many existing host-based IDSs employed on individual host machines in the target information system. We refer to it as ISA-IDS. It is designed to be cross-platform and distributive. Only an event data collector needs to be installed in each of the host machines in the target information system to passively monitoring activities and collect activity data. The users of a host machine may not notice the event data collector at all. The event data collectors send only relevant event information to the centralized IDS server for farther processing. The centralized IDS server in turn calls on the individual technique servers to process the event data stream one by one. Each of the individual technique servers uses a different intrusion detection algorithm and returns a test value indicating how the specific individual technique considers the input event as normal or intrusive. The test values from all the individual technique servers are combined together in the centralized IDS server using a fusion algorithm. Using several individual technique servers ensure that we cover different kinds of intrusion detection techniques and offset one technique's shortcomings with another one's advantages, making the intrusion detection more accurate and robust. New intrusion detection algorithms can also be added into ISA-IDS through deploying more individual technique servers. We have tested this prototype using a large portion of DARPA (Defense Advanced Research Projects Agency)'s 1998 evaluation data (Lippmann, 2000).

This chapter describes the architecture of ISA-IDS and implementation details. We first review some of the existing IDSs. We then present the collection process and characteristics of the DARPA evaluation data that we have used to test a prototype of ISA-IDS. We also describe the implementation of the prototype and individual techniques in this prototype, and summarize the testing results.

By illustrating the design of the ISA-IDS system architecture and the concepts underlying our intrusion detection and information fusion techniques, we demonstrate how different intrusion detection techniques work together in an IDS to detect intrusions and what the IDS requires from the target information system. We hope that this will help people (i.e., SSAs, executives, managers, and others) gain a better understanding of the current state-of-the-art of IDSs and make better decisions to protect information systems in their organizations.

II. RELATED WORK

There are surveys on existing host-based, network-based and router-based IDSs (Axelsson, 1998; Debar et al., 1999). In this paper we focus on host-based IDSs. Host-based IDSs monitor activities occurring on individual host machines in an information system through computer audit data. A detailed description of computer audit data is provided in the next section.

Table 1-1 summarizes the characteristics of some host-based IDSs. Those IDSs are distinguished with regard to the intrusion detection technique, the information fusion technique, and the distribution of detection functions.

Table 1-1. Characteristics of existing host-based IDSs.

IDS	**System Architecture**	**Intrusion Detection**	**Fusion**
CI	Centralized	Anomaly detection	
CSM	Distributed on target hosts	Signature recognition	
DPEM	Distributed on target hosts	Anomaly detection	
COAST-	Distributed on target hosts		

EIMDT			
EMERALD	Distributed on target hosts and security servers	Signature recognition Anomaly detection	
GrIDS	Centralized	Anomaly detection	
Haystack	Centralized	Signature recognition Anomaly detection	Fixed weights
Hyperview	Centralized	Signature recognition Anomaly detection	
IDES/NIDES	Centralized	Signature recognition Anomaly detection	
IDIOT	Centralized	Signature recognition	
NADIR	Centralized	Signature recognition	
RIPPER	Centralized	Signature recognition Anomaly detection	
USTAT	Centralized	Signature recognition	

Note: a blank cell in the table indicates that either information or technology is not available.

Intrusion detection techniques generally fall into two categories: signature recognition and anomaly detection. Signature recognition (also called "misuse detection" in some literature) techniques identify and store signature patterns of known intrusions, match activities in an information system with known patterns of intrusion signatures, and signal intrusions when there is a match. Signature recognition techniques are efficient and accurate in detecting known intrusions, but cannot detect novel intrusions whose signature patterns are unknown. Anomaly detection techniques establish a profile of a subject's normal activities (a norm profile), compare observed activities of the subject with its norm profile, and signal intrusions when the subject's observed activities differ largely from its norm profile. The subject may be a user, file, privileged program, host machine, or network. Anomaly detection techniques can detect both novel and known intrusions if they demonstrate large deviations from the norm profile. Since anomaly detection techniques signal all anomalies as intrusions, false alarms are expected when anomalies are caused by behavioral irregularity instead of intrusions. Hence, signature recognition techniques and anomaly detection techniques are often used together to complement each other.

When both a signature recognition technique and an anomaly detection technique are used in an IDS to monitor the same observed activities in an information system, the two different techniques may produce different evaluation results. An information fusion technique is required to combine different results from different intrusion detection techniques on the same observed activities into a composite score for an overall assessment of intrusion likelihood with the observed activities. Very few IDSs incorporate an information fusion technique. The existing information fusion techniques for intrusion detection are based on mainly a fixed weight vector that assigns a fixed weight to the result from an individual intrusion detection. The fixed weight reflects the relative importance of an individual intrusion technique in producing the overall assessment of intrusion likelihood.

Existing IDSs also differ in their architecture to support various methods of distributing intrusion detection functions. Two common architectures are: centralized and distributed. A centralized architecture distributes intrusion detection functions usually on a security server. A distributed architecture usually distributes intrusion detection functions to security modules running on the target host machines in an information system. Sometimes, security servers are also used in a distributed architecture to coordinate security modules and perform further analysis.

Each IDS listed in Table 1-1 is briefly described below to give some concrete pictures of how IDSs work. The Purdue University COAST (Computer Operations, Audit, and Security Technology) Laboratory's Enhanced Intrusion and Misuse Detection Techniques program (COAST-EIMDT) (Balasubramaniyan et al., 1998) relies on multiple independent entities, called autonomous agent, working collectively. Autonomous agents perform certain security monitoring function at each host. They are independently running entities, i.e., their execution is scheduled only by the operating system, and not by other processes. Agents may or may not need data produced by other agents to perform their work and may receive high-level control commands from other entities. The agents can be added and removed from a system without altering other components and without restarting the IDS and can be tested on their own before introducing them into a more complex environment. The results collected from the agents are organized into a hierarchical manner to ensure scalability of the system. Hence, COAST-EIMDT uses a distributed architecture with intrusion detection functions on target hosts. Although COAST-EIMDT does not specify intrusion detection techniques that it may use, any intrusion detection can fit into this architecture to carry out the intrusion detection functions.

IDES (Intrusion Detection Expert System) (Lunt et al., 1992) is a real-time intrusion detection expert system, which has now been merged into NIDES (Jagannathan et al., 1998), the Next generation IDES. The audit data collected in the hosts is securely transmitted to NIDES via the network and processed to provide real-time response, The statistical engine of IDES/NIDES verifies each new audit record against the norm profiles for both the subject and the group of subject it belongs to. It also verifies each session against the norm profile of sessions when the session completes. IDES/NIDES also consists of an expert system that keeps signature patterns of known intrusions. Hence, IDES/NIDES uses a centralized architecture of intrusion detection functions, and employs both signature recognition and anomaly detection techniques. When different measures of the same observed activities are used to detect intrusions, IDES/NIDES proposes a squared sum of the results from individual measures to generate a composite score.

IDES/NIDES does not address the combination of the results from its statistical engine and the expert system for signature recognition.

GrIDS (Graph based Intrusion Detection System) (Chen et al., 1996) employs data source modules that runs on each host and reports relevant information to graph engines that build a graph representation of activity in the network and compare the graph with the norm profile for detecting intrusions. Hence, GrIDS uses a centralized architecture, and employs mainly an anomaly detection technique for intrusion detection.

NADIR (Network Anomaly Detector and Intrusion Reporter) (Hochberg et al., 1993) performs distributed data collection by employing the existing service nodes in Los Alamos National Laboratory's Integrated Computer Network (ICN) to collect audit information, which is then analyzed by a central expert system that compares the summary of the weekly audit data with rules in the expert system describing violations of security policies and suspicious activities. Hence, NADIR uses a centralized architecture, and employs a signature recognition technique for intrusion detection.

In the peer-based IDS described in (White et al., 1996) Cooperative Security Managers (CSM) are employed to perform distributed intrusion detection that does not need a hierarchical organization or a central coordinator. Each CSM performs as a local IDS for the host where it runs, but can additionally communicate with other CSMs and exchange information about users moving through the network and detect suspicious activities based on signature recognition. Hence, CSM uses a distributed architecture of intrusion detection functions on the target host machines, and employs a signature recognition technique for intrusion detection.

The IDS based on computer immunology (CI) emulates the biological immune systems to some extent by employing a set of "self" patterns of event sequences to detect "non-self" at various key locations in the information system (Forrest et al., 1997). CI uses a centralized architecture, and employs an anomaly detection technique for intrusion detection.

The EMERALD project (Porras & Neumann, 1997) proposes a distributed architecture for intrusion detection that employs entities called service monitors that are deployed to hosts and perform monitoring functions. It also defines several layers of monitors for performing data reduction in a hierarchical fashion. Hence, EMERALD uses a distributed architecture of intrusion detection functions on the target host machines and some security servers, and employs both a signature recognition technique and an anomaly detection technique from IDES/NIDES. EMERALD does not address the fusion of results from different intrusion detection techniques for the same observed activities.

Hyperview (Debar et al., 1992) employs an artificial neural network component and an expert system component for intrusion detection. The expert system component monitors audit trails for known signatures of intrusion. The artificial neural network component adaptively learns the behavior of a user and raises alarms when the audit events deviate from the already learned behavior. Hyperview connects the artificial neural network to two expert systems. The first one monitors the operation and the training of the network and evaluates its output. The other scans the audit trail for known patterns of intrusions, and together with the output from the first expert system forms an opinion if it should raise an alarm. Hence, Hyperview employs both a signature recognition technique and an anomaly detection technique for intrusion detection, and uses a centralized architecture.

USTAT (Unix State Transition Analysis Tool) (Ilgun, 1993) is a prototype implementation of the state transition analysis approach to intrusion detection. The state transition analysis takes the view that the computer initially exists in a secure state, but as a result of a number of penetrations - modeled as state transitions - it ends up in a compromised target state. That is, state transitions are used to describe signature patterns of known intrusions. USTAT uses a centralized architecture, and employs only a signature recognition technique.

DPEM (Distributed Program Execution Monitoring) (Ko et al., 1997) focuses on the correct security behavior of the system, i.e., a security privileged application that runs on the system. DPEM reads the security specifications of acceptable behavior of privileged UNIX programs, and checks audit trails for security violations. The DPEM prototype monitors programs executed in a distributed system by collecting execution traces from the various hosts, and where relevant, distributing them across the network for processing. Hence, DPEM uses a distributed architecture, and employs an anomaly detection technique for intrusion detection.

IDIOT (Intrusion Detection In Our Time) (Kumar & Spafford, 1995) was developed at COAST (now the Center for Education and Research in Information Assurance and Security - CERIAS) in Purdue University. It employs Colored Petri-nets (CP-nets) for signature based (pattern matching) intrusion detection and suggests a layered approach for applying signature-based techniques. The patterns (signatures) are written in an ordinary textual language and then parsed, resulting in a new pattern-matching engine. This engine can then be dynamically added to an already running IDIOT instance via the user interface. The user can even extend IDIOT to recognize new audit events. Hence, IDIOT uses a centralized architecture, and employs a signature recognition technique for intrusion detection.

RIPPER (Lee et al., 1999) uses data mining for automatic and adaptive construction of intrusion detection models. It utilizes auditing programs for extracting extensive set of features to describe each network connection or host session, and applies data mining programs to learn rules that accurately capture the behavior of intrusions and normal activities. These rules can then be used for signature detection and anomaly detection. The RIPPER framework implements several algorithms. The user can pre-process audit data to provide the algorithms with a representation on a natural level. The user then gives RIPPER the task of automatically mining patterns of abusive or normal behavior. Hence, RIPPER uses a centralized architecture, and employs a data mining technique for both signature recognition and anomaly detection.

Haystack was developed as an audit data reduction and intrusion detection tool (Smaha, 1988). It collects audit data from a target host machine, and performs primitive statistical analysis to detect deviations of observed activities from normal activities and to compare observed activities with known intrusions. Haystack is the only IDS in Table 1-1 that incorporates an information fusion technique. Haystack calculates a weighted intrusion score by summing the results from multiple intrusion detection components. Fixed weights are assigned. Hence, Haystack uses a centralized architecture, and employs both a signature recognition technique and an anomaly detection technique for intrusion detection along with the fixed-weight technique for information fusion.

Table 1-1 does not include virus scanners and monitors. Most virus scanners and monitors are based on signature recognition techniques to capture signatures of viruses and use those signatures to detect viruses. Those virus scanners and monitors can also be considered as IDSs with focus on data in the file system and other media rather than computer audit data. Because virus scanners and monitors monitor data that are different from computer audit data monitored by host-based IDSs, virus scanners and monitors can be included in host-based IDSs as additional components to produce separate intrusion reports.

Many of the IDSs in Table 1-1 are still in a research prototype stage. Those commercial IDSs rely mainly on signature recognition techniques with the inherent limitation of detecting novel intrusions (Debar et al., 1999). Some of the existing IDSs, such as Hyperview and EMERALD, incorporate both signature recognition techniques and anomaly detection techniques. However, if both signature recognition techniques and anomaly detection techniques are used together to monitor the same activities in an information system, we should expect different results from different intrusion detection techniques. The fusion of different results into a composite index of intrusion warning is necessary.

Hence, in addition to signature recognition techniques and anomaly detection techniques, our ISA-IDS also includes the function of information fusion. Moreover, the signature recognition techniques and the anomaly detection techniques in the ISA-IDS have significantly advanced the state-of-the-art of intrusion detection by overcoming problems with the existing intrusion detection techniques in performance and scalability (Ye et al. 1, 2000; Ye et al. 2, 2000; Ye & Li, 2000; Ye et al. 3, 2000; Ye, 2000). The main driver of developing the ISA-IDS lies in the advantage of these intrusion detection techniques. Currently, the ISA-IDS focuses on computer audit data, and uses a distributed architecture. ISA-IDS places little overhead on the target information system, and uses independent host machines (security servers) to perform intrusion detection. ISA-IDS will have the capability of processing network traffic data in the future. The following sections provide more details in the design, implementation and testing of ISA-IDS.

III. DATA SOURCE

ISA-IDS is a distributed host-based IDS that has been designed to collect audit event data from many hosts. In the prototype implementation of ISA-IDS we have simplified the data collection process by using the audit data from the Defense Advanced Research Projects Agency

(DARPA). This has been done to keep our focus towards implementing the centralized IDS server and the individual technique servers. In the next stage of research we will develop the event data collectors.

The Information Systems Technology Group of MIT Lincoln Laboratory (LL), under the DARPA Information Technology Office and Air Force Research Laboratory (AFRL) sponsorship, collected network traffic data and computer audit data containing both normal and intrusive activities from a simulation network, and developed a guideline for evaluating existing intrusion detection systems in 1998 (Lippman et al., 2000). The simulation network architecture to collect the network sniffed traffic data and audit data is shown in Figure 1-1. Figure 1-1 shows the points where network traffic data from a Sun OS sniffer and computer audit data from the Solaris Basic Security Module (BSM) are collected. The ISA-IDS prototype uses only the Solaris Basic Security Module (BSM) audit data from this 1998 DARPA-LL data set.

For this 1998 DARPA-LL data set, normal activities were simulated to resemble normal activities occurring in a real computer network at Air Force (Lippman et al., 2000). A large amount of web, telnet, and mail traffic was generated between the inside host machines and the outside host machines and web sites. Many user automata of various types such as secretaries, programmers and managers were also used to generate normal activities. Statistical profiles of various user types were used to simulate normal activities. These statistical profiles consider information such as the occurrence frequency of different UNIX commands, typical login times and session durations, and so on. Human actors were also used to perform more complex tasks such as upgrading software, adding users, remotely accessing programs, and performing other system administrative tasks. Those simulated normal activities have created the "white noises" background.

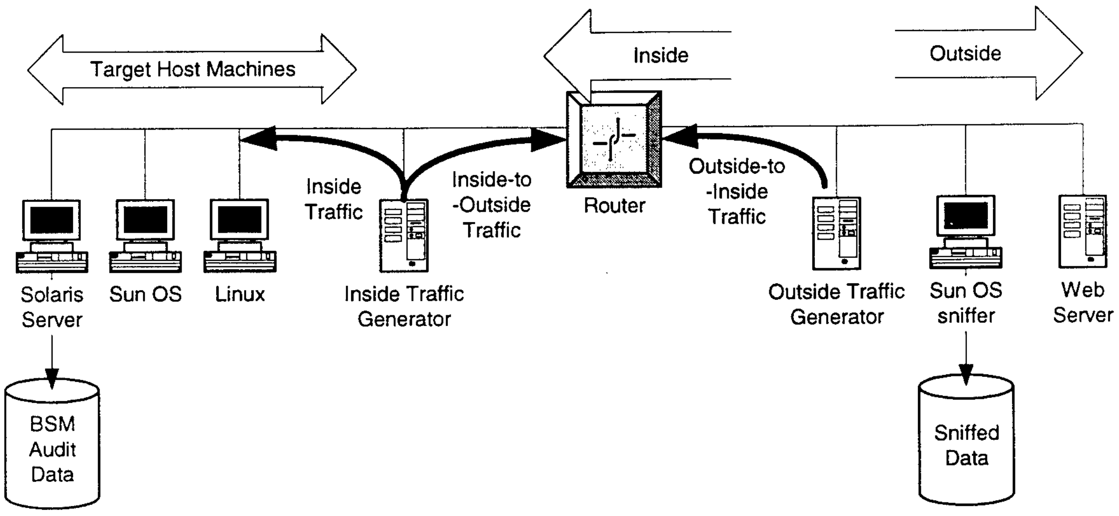


Figure 1-1. The simplified offline simulation network architecture for collecting network sniffed and audit data.

While simulating normal activities in the simulation network, intrusions have also been simulated on the "white noise" background. Simulated intrusions fall into four categories: denial of service attacks to disrupt a host or network service, remote-to-local attacks where an attacker without an account on a victim machine attempted to gain local access, user-to-root attacks where a local user attempted to gain privileges of the UNIX root user, and probe-scan attacks using programs to probe or scan a network for gathering information such as a topology of the network. More details about the simulation of normal and intrusive activities for the 1998 DARPA-LL data can be found in (Kendall, 1999; Lippmann et al., 2000; Northcutt, 1999).

Normal and intrusive activities occurring on a Sun Solaris machine in the simulation network have been recorded by the Solaris auditing facility, BSM, producing the computer audit data with a mixture of normal and intrusive activities. BSM audit data consist of a sequence of audit files. Each audit file consists of many audit records. Each audit record captures an auditable event. Each audit record includes a sequence of audit tokens, each of which contains specific information about the attributes of the event. By an "event" we point to a particular activity in the underlying computer system at a certain time along with the information pertaining to that activity. "Attributes" of an event are the variables that contain information about such activity.

There are twenty-five different audit tokens. However, most audit records contain only a few tokens such as the header token with such information as an event ID indicating the type of audit event, the time when the record was created and so on, the subject token with such information as the user ID, the group ID, the session ID and so on, and a return token indicating the return status of the event.

The BSM of Solaris monitors each event on a host machine and generates a sequence of audit records. Auditable events are generated by system calls used by the kernel of the operation system or by application programs. BSM defines 284 different types of audit events. There are thousands of commands in Solaris UNIX. The audit events are more close to the kernel of the operating system. Hence, the type of event is more representative than the actual command sequences used. For example, we can use any text editor, such as, vi, ed, pico to edit a file, but most of the time the audit event stream will contain the following event types: AUE_EXECVE, AUE_OPEN_R, AUE_ACCESS, AUE_STAT.

The 1998 DARPA-LL data are provided in two parts – the training data set and the testing data set. The training data set is provided for developing an IDS system and configuring its parameters. The testing data set is provided for testing the performance of the trained IDS system. There are more than 300 instances of 38 different intrusions in the training and testing data. Using the provided description of intrusions in the training data from the MIT Lincoln Laboratory, we can label each audit record and its corresponding audit event in the training data set as either normal or intrusive along with the intrusion name. The training data set covers intrusive activities occurring in the midst of normal activities for seven weeks. Another two weeks of network sniffed traffic data and BSM audit data containing both normal and intrusive data are provided as the testing data. This data set contains all those intrusions that were in training set and also new type of intrusions. The data records in the testing data set are not labeled. The IDS are allowed to scan though this data set only once, and the IDS has to label, using its algorithms, each data record as normal or intrusive.

The ISA-IDS prototype conforms to this specification from the MIT Lincoln Laboratory for evaluating intrusion detection systems. The Centralized IDS Server of the prototype ISA-IDS scans through the training data set to set the parameters of the techniques we have integrated in the system through the Individual Technique Servers. The Centralized IDS Server then takes the testing data set, passes each audit data record to the Individual Technique Servers, fuses their results, and labels each audit data record as either normal or intrusive.

Since no label is provided for audit events in the testing data set of the 1998 DARPA-LL data, we do not know the ground truth of these audit events and thus cannot evaluate the performance of the ISA-IDS prototype with the ground truth. Therefore, we cannot use the testing data set for testing the ISA-IDS. Instead, we drew our two-day training data and two-day testing data from the seven-week training data set of the 1998 DARPA-LL data. Two-day data are used for training and testing to reduce the amount of time required to complete the training and testing. In the seven weeks of training data, there are in total 35 days of data. We picked four days of data as a representative of these 35 days of data. We wanted to pick two days where there are not many intrusions and two days where there are many intrusions. According to this criterion, we chose week-1, Monday data as day-1 data, week-4, Tuesday data as day-2 data, week-4, Friday data as day-3 data and week-6, Thursday data as day-4 data. These days are nonconsecutive, comes from different weeks and each are of different weekday. We changed the day marks of day-2 data, day-3 data, and day-4 data so that four days become consecutive days.

IV. ISA-IDS ARCHITECTURE

This section elaborates the ISA-IDS system architecture. Figure 1-2 sketches the system architecture. It consists of one Centralized IDS Server (CIS) that collects the audit events from the hosts that it is monitoring. We deploy an Event Data Collector (EDC) in each of the host machines. The Event Data Collector collects the BSM audit data from the host and sends audit events to the Central Event Collector (CEC) of the Centralized IDS server. The Central Event Collector in turn sends them through the Event Dispatcher to the Individual Technique Servers (ITS) that use different intrusion detection algorithms. Each Individual Technique Server provides its response to the Centralized IDS Server by calculating the intrusion warning (IW) level for that event. After getting back the results, the Centralized IDS Server fuses the different IW levels from the Individual Technique Servers for the same event and produces a composite IW level for that event.

The Centralized IDS Server acts like a centralized control center of the entire network system. Its main tasks are the followings in order (Figure 1-3 shows the data flow diagram of the ISA-IDS):

1. Collect audit event data from the hosts through the Central Event Collector;
2. After data pre-processing, dispatch the audit event stream to the Individual Technique Servers through the Event Dispatcher;
3. Collect the IW levels for events from the Individual Technique Servers;

4. Fuse the IW levels and produce a composite binary IW level (1 for intrusive and 0 for normal) for each event;
5. Label the input event as intrusive or normal according to the composite IW level and produce an alert signal if required.

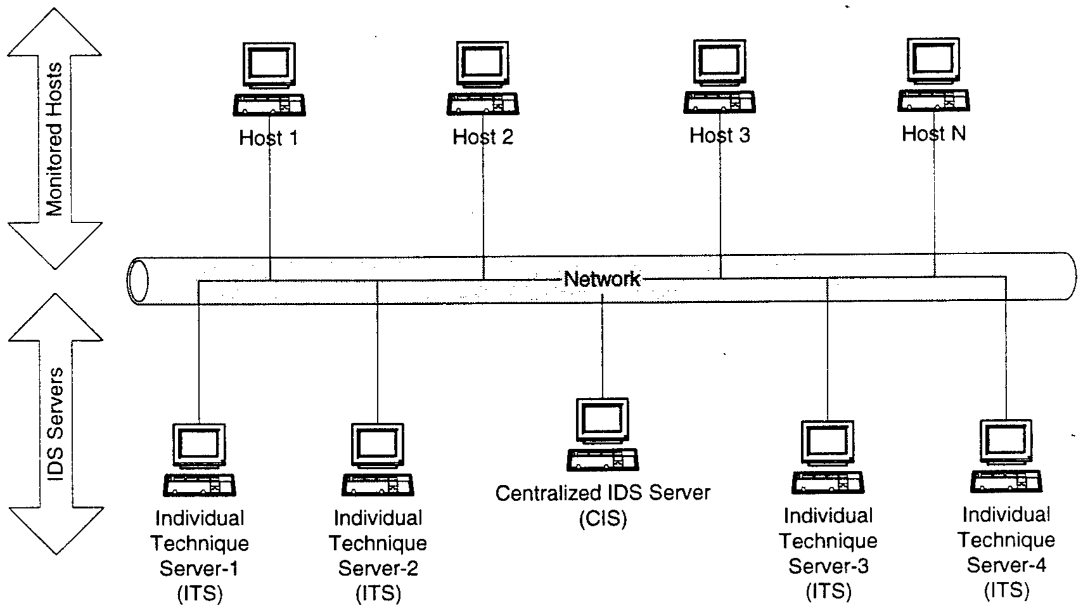

Monitored Hosts
Host 1
Host 2
Host 3
Host N
Network
IDS Servers
Individual Technique Server-1 (ITS)
Individual Technique Server-2 (ITS)
Centralized IDS Server (CIS)
Individual Technique Server-3 (ITS)
Individual Technique Server-4 (ITS)

Figure 1-2. System architecture of the ISA-IDS.

Therefore, after deploying the Event Data Collectors in the host machines, setting up the Centralized IDS Server and the Individual Technique Servers and starting to run the system, the SSA only has to check the Centralized IDS Server from time to time for any possible intrusion alert.

The SSA may have to manually inspect all the alarms raised by the ISA-IDS to figure out the cause of the intrusion. Sometimes the ISA-IDS may produce a warning signal for an event that is not in fact intrusive - this is an example of a false alarm. Sometimes the ISA-IDS may not produce a warning signal for an event that is in fact intrusive – this is an example of a miss. The lower the false alarm rate and the miss rate, the better the ISA-IDS.

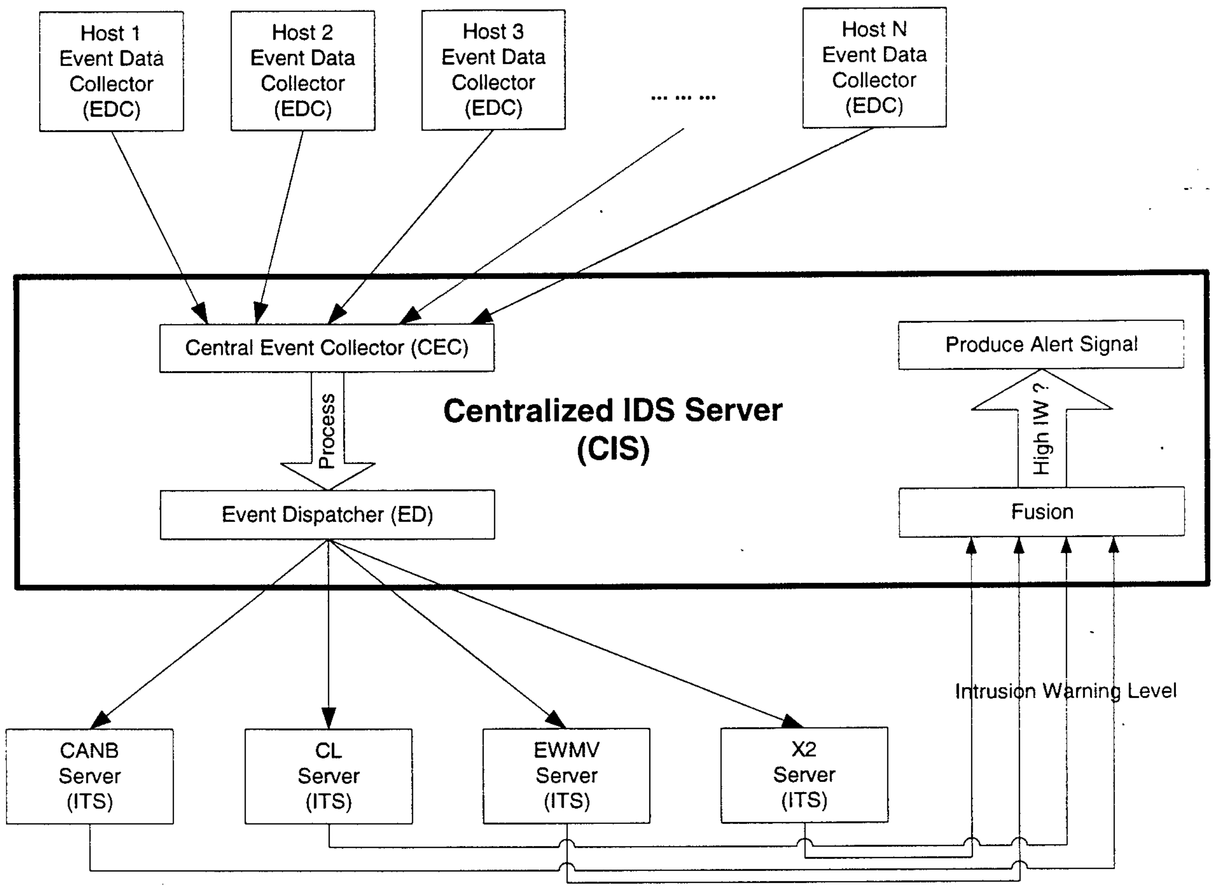

Host 1 Event Data Collector (EDC)
Host 2 Event Data Collector (EDC)
Host 3 Event Data Collector (EDC)
......
Host N Event Data Collector (EDC)
Central Event Collector (CEC)
Process
Centralized IDS Server (CIS)
Event Dispatcher (ED)
Produce Alert Signal
High IW ?
Fusion
Intrusion Warning Level
CANB Server (ITS)
CL Server (ITS)
EWMV Server (ITS)
X2 Server (ITS)

Figure 1-3. Data flow diagram of the ISA-IDS.

At the beginning we need to train the Individual Technique Servers using the training data. Initially, a training data set containing normal and intrusive activities, such as the 1998 DARPA-LL data, can be collected and used for training. After the initial training the system can use the audit data collected from the hosts in the monitored information system. Since the Individual Technique Servers have already been trained, they can produce intrusion labels for each of the events in the event stream. During off-peak hours, say after midnight, the data collected during the day can be used for updating or retraining the system parameters of each of the techniques inside Individual Technique Servers. The retraining can be scheduled per day, per week or per month basis as desired by the system administrator. Currently the ISA-IDS prototype uses the training part of the 1998 DARPA-LL audit data set to train the Individual Technique Servers and labels the testing data set during testing.

For an Individual Technique Server supporting an anomaly detection technique, the training of the technique requires only computer audit data of normal activities. Because each audit record in the training data set of the 1998 DARPA-LL data can be labeled as normal or intrusive, we can filter the training data set by its label so that we have computer audit data of only normal activities to train this anomaly detection technique. For an Individual Technique Server supporting a signature recognition technique, the training of the technique requires computer audit data of both normal and intrusive activities. The training data set of the 1998 DARPA-LL data can be used directly without filtering to train the technique. Details of the Individual Technique Servers are provided in the following sections.

In an environment where there is an absence of data from sources where intrusions are known and characterized, two approaches can be taken to train the Individual Technique Servers for signature recognition techniques. In the first approach, the Individual Technique Servers for signature recognition techniques in the ISA-IDS can be disabled at first. Therefore, the ISA-IDS relies on the Individual Technique Servers to label incoming audit records. When intrusions are detected, the Individual Technique Servers for signature recognition techniques can use the labeled audit records to discover the signatures of those known intrusions and thus get trained. In the second approach, because many intrusion scenarios that happened in the past are known, those intrusion scenarios can be simulated just as what LL did to create computer audit data of intrusive activities. These two approaches can be used alone or together.

In the past few years we have investigated several techniques for intrusion detection covering both anomaly detection and signature recognition. The detailed description and performance results of the techniques can be found in (Ye et al. 1, 2000; Ye et al. 2, 2000; Ye & Li, 2000; Ye et al. 3, 2000; Ye, 2000). Four of these techniques have been incorporated into the ISA-IDS prototype. The techniques are named - CANB for Canberra metric, CL for Clustering algorithm, EWMV for Exponentially Weighted Moving Variance technique and X2 for Chi-square test. These techniques and their implementation are briefly described in the next section.

Since the amount of data to be processed is huge, and each of the techniques needs a considerable amount of disk space and processing power we have used a separate machine for each of the techniques. Therefore one technique corresponds to one Individual Technique Server. The Centralized IDS Server must have a very high processing power and very large amount of disk space to store all the incoming events and results produced. Separating the techniques into different Individual Technique Servers enables us to comfortably plug-in new kinds of intrusion detection techniques with minor modification in the event dispatcher and fusion technique of the Centralized IDS server. Note that if desired, the Centralized IDS Server and Individual Technique Servers can reside on the same machine rather than on different machines.

V. PROTOTYPE IMPLEMENTATION AND INTRUSION DETECTION TECHNIQUES

In the prototype implementation we have used the 1998 DARPA-LL BSM audit data and have not yet incorporated the host Event Data Collectors. Except the Event Data Collectors in the hosts and the Central Event Collector in the Centralized IDS Server we have implemented all other components. We have implemented the Centralized IDS Server and the Individual Technique Servers with Visual C++™ and for communication between them over the network we have used the Orbix™ implementation of CORBA. Following CORBA standards, the Individual Technique Servers have been implemented as CORBA servers and the Centralized IDS Server has been implemented as a CORBA client. The Centralized IDS Server scans through the BSM audit events and dispatches the event objects to the servers through the Event Dispatcher by calling appropriate server services that the Individual Technique Servers implement. All the Individual Technique Servers have been implemented in a uniform way so that new Individual Technique Servers can be easily plugged in.

We can model the entire network as a system and the hosts that we are monitoring as independent subsystems. Since the 1998 DARPA-LL MIT data contains BSM audit data only from a single host machine with the Solaris operating system, there is currently only one subsystem in the prototype. When there are audit data from a network of multiple host machines, the ISA-IDS can also set up the system and multiple subsystems.

We have analyzed the incoming BSM audit events from two perspectives - "Event Type" (Ye et al. 1, 2000; Ye et al. 2, 2000) and "Process Model" (Ye & Xu, 2000). Note that the SSA can select either of the two methods to use for an information system. Figure 1-4 highlights the difference between these two methods through the simplified class diagram of the prototype.

In the Event Type method, we look at the type and time of event occurrence and pass that information to the Individual Technique Servers. We ignore other information related to that event. This way of analysis is much simpler and faster but ignores details of each event. For example, if the event is related to a file that a user is using, Event Type analysis does not care about which user is using which file. Its analysis is based entirely on the sequence of the type of events.

According to Process Model, we look more into the events and find out whether each event is related to Login or File or User or any of the Daemon objects. An event can involve more than one of these object classes. For example, if user A accesses file B - it corresponds to an event related to a user object (A) and also to a file object (B). Therefore, the Process Model method of data analysis not only considers the type of the event but also the actual source of the event and gives more information to the Individual Technique Servers. It is more complex than the Event Type, and therefore requires much more processing power.

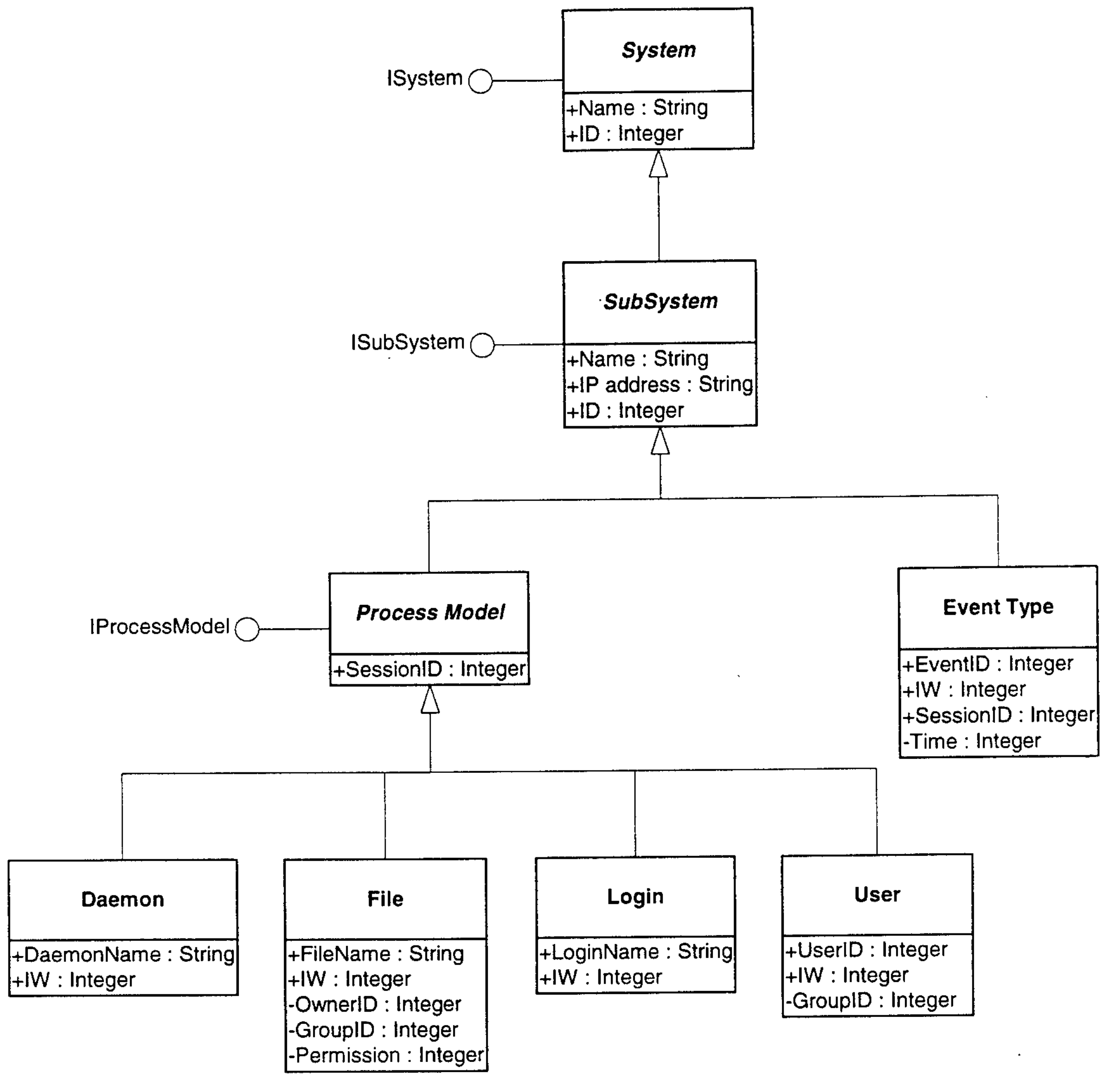


Figure 1-4. Simplified class diagram of the ISA-IDS.

In the Process Model method, depending on the objects (Daemon, File, Login, User) associated to an event, we create those objects involved in the event. In the Event Type method, we have only one type of object to create – the subsystem for the single host machine. After filling up their attributes properly, the Event Dispatcher of the Centralized IDS Server sends those objects to the Individual Technique Servers. Each Individual Technique Server scans its database looking for the previous occurrence of that object and retrieves its profile and sets the Intrusion Warning (IW) level ($0 \leq IW \leq 1$, higher value indicates more intrusiveness) according to the algorithm of that Individual Technique Server. The Individual Technique Server then returns the object back to the Centralized IDS Server with the IW level filled in. After collecting the IW levels from all the Individual Technique Servers, the Centralized IDS Server feeds these into the fusion component to produce a composite IW level. If the level is 1, a signal is generated notifying the SSA to take farther action. Figure 1-5 illustrates the procedures taken in the Process Model method or the Event Type method to calculate the IW value.

The rest of the section describes the four Individual Technique Servers we have incorporated - CANB, X2, EWMV and CL.

Each object associated with an event has several attributes, which we refer as variables. For example, in the Event Type method, we have 284 event types and therefore there are 284 variables, numbered from 1 to 284. In the Process Model method, the number of variables depends on the type of the objects (Daemon, File, Login, or User) associated with the event. We have defined the variables for each of the objects after thoroughly studying the characteristics of each object class.

The training consists of several phases, and is slightly different from one Individual Technique Server to another, though they follow the same procedure. In phase I, the Event Dispatcher of the Centralized IDS Server takes the events chronologically one by one and creates associated objects and passes them to the Individual Technique Servers. For each event on each object, an Individual Technique Server calculates an observation value of the variables for the event on the object, updates the training parameters for the Individual Technique Server, and stores the observed value (we refer to it as *X* value) and the training parameters. In phase II, the

Individual Technique Server takes the *X* value stream and computes the *T* value according to the equation of a specific technique along with the average and standard deviation of the T value stream, if needed (for CANB, X2 and CL). The average and standard deviation of the T value stream are used to determine the control limits of the *T* value for that technique.

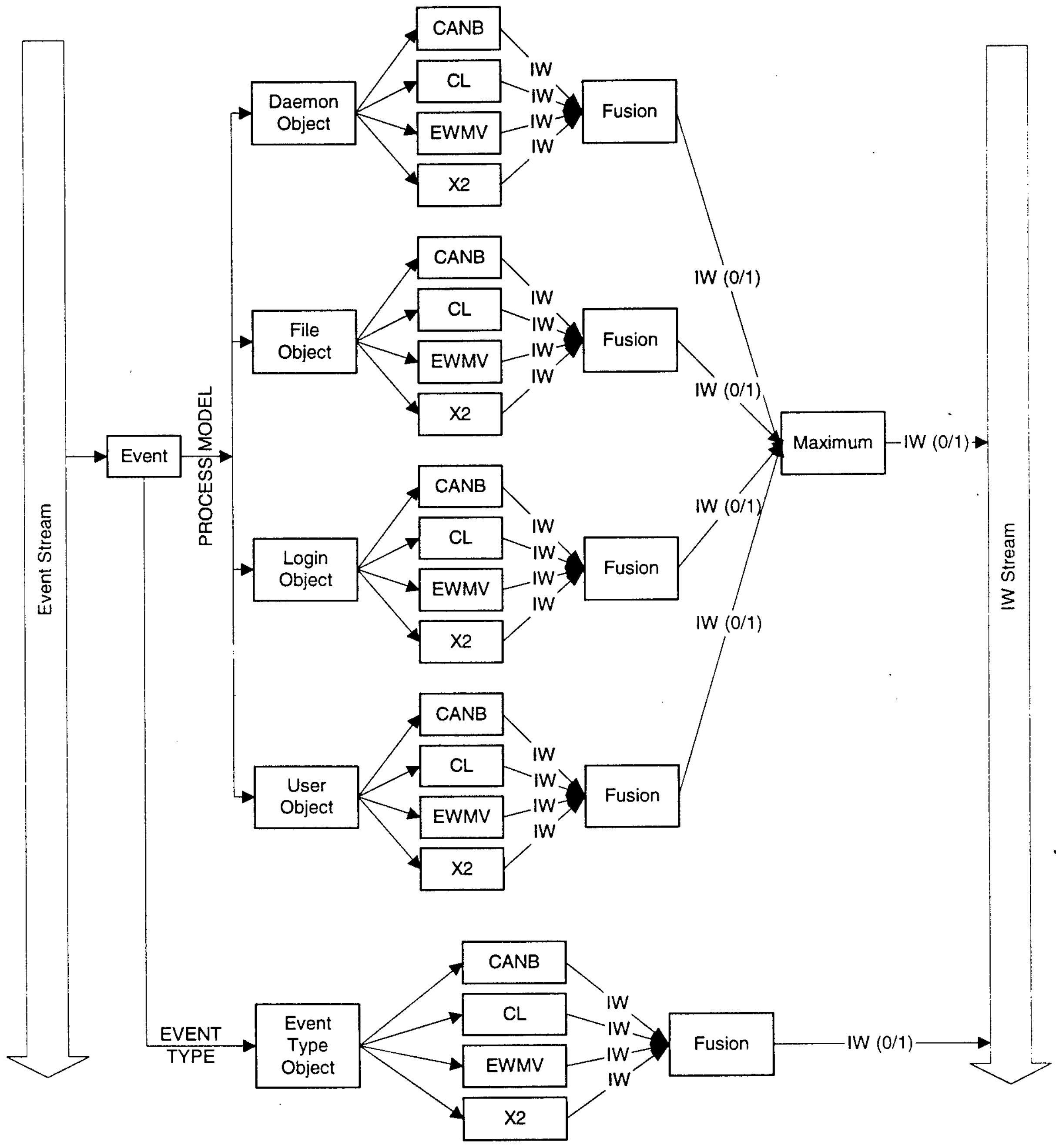


Figure 1-5. IW calculation in Process Model and Event Type methods.

In testing, the X value and the T value are computed for each audit event. The control limits of the T value from phase II of the training are then used to transform the T value into the IW value that indicates whether or not to signal an intrusion.

Now we briefly discuss the algorithm in each of the Individual Technique Servers during training and testing. During testing only X values, T values and IW levels are calculated. All the training parameters are calculated in training.

The CANB is an anomaly detection technique. It builds a norm profile in training using only audit data of normal activities. The norm profile is then used in testing to determine the IW level of each audit event. During training and testing, the CANB technique uses the Exponentially Weighted Moving Average (EWMA) method (Ryan, 2000) to compute an observed value – the X value - for the event on an object. X is a vector consisting of multiple variables. The observed value of a variable indicates the occurrence frequency of activity that the variable represents. For example, in the Event Type method, the observed value of the 284 variables is computed for each audit event. The observed value of variable i represents the occurrence frequency of event type i in the recent past. In training, only audit data of normal activities are used to build the norm profile. In phase I of training, the X values for all the audit events in the training data set and the mean vector of the X values for those audit events are computed. Phase II goes over the X value stream from phase I, and transforms the X value for the n^{th} event into the Canberra distance metric from the mean vector the X values to produce the T value for this event. Hence, the T value measures the distance of the X value for this event to the mean vector of X according to the Canberra distance definition. Meanwhile, the mean and standard deviation of the T values are updated incrementally by taking into account the T value for this event. After the last event in the training data set, the incrementally updated mean and standard deviation of the T values become the mean and standard deviation of the T values for all the training data, $\overline{T}$ and S_T. This mean and standard deviation of the T values are used to determine the control limits as follows: $\left[\overline{T} - CLF * S_T, \overline{T} + CLF * S_T\right]$ which indicates the range of the T values for normal activities, where CLF is a control limit factor, and is usually set to 2 or 3.

In testing, the X value and the T value are computed for each audit event. If the T value falls outside the range of the control limits (the range of normal activities), the IW level is set to 1 (a signal indicating an intrusion); otherwise, the IW level is less than 1 (no signal). The specific IW level depends on how close the T value is to the center of the range of the control limits (how close the T value is to the mean vector of T).

The X2 technique is an anomaly detection technique similar to the CANB technique. The two techniques differ only in the definition of the distance metric that is used to compute the T value for an event. For the X2 technique, the chi-square distance metric is used.

Both the CANB technique and the X2 technique monitor the frequency distribution of multiple variables (i.e., for various event types in the Event Type method) and detect anomalies as intrusive. SSA can select either of the two techniques when running the ISA-IDS. Because these two techniques are similar, there is no need to select both.

The EWMV technique is an anomaly detection technique. It uses the Exponentially Weighted Moving Variance technique to monitor the event intensity for an object. It is designed to detect significant changes in the event intensity. Many denial-of-service intrusions manifest through changes in the event intensity. The only information that the EWMV technique uses of each event is the time of event occurrence. Therefore the EWMV technique uses only one variable for the *X* value and does not calculate the *T* value. The EWMV technique uses a time decaying method to calculate the observation value reflecting the intensity of events in the most recent past (the frequency count of events per time unit). The exponentially weighted moving average (EWMA) and the exponentially weighted moving variance (EWMV) are then calculated to set the control limits as follows: [EWMA – CLF*EWMV, EWMA + CLF*EWMV].

In testing, the IW level is computed by comparing the observed event intensity with the control limits. If the *X* value for an event falls outside the control limits, the IW level is set to 1 (a signal for intrusion); otherwise, the IW level is less than 1 (no signal).

The CL technique is a signature recognition technique that learns intrusion signatures from the training data of both normal and intrusive activities, and uses the learned signatures to detect intrusions in testing. The CL technique employs a clustering and classification algorithm to cluster audit events in the training data set and to classify each audit event in the testing data set into normal or intrusive. The CL technique uses the same method of computing the observation value – the X value – as in the CANB and X2 techniques. The variables associated with an event are considered as dimensions. Hence, an observation is a data point in a multi-dimensional space. In training, the CL technique groups data points in the training data set into normal and intrusive clusters. In testing, the CL technique uses the cluster structure to determine the IW level of an event by examining whether the observation value for the event is close to normal clusters or intrusive clusters.

In the prototype of the ISA-IDS, we use either the X2 technique or the CANB technique to fuse the IW levels from the Individual Technique Servers into a composite IW level. When using the X2 technique or the CANB technique for information fusion, the IW levels from the Individual Technique Servers produce the *X* value consisting of multiple variables for those individual Technique Servers respectively. That is, the information fusion is based on the norm profile of the Individual Techniques Servers' outputs for normal activities in training. In testing, if the results from the Individual Technique Servers for an event deviate significantly from the norm profile, the information fusion technique produces a composite index triggering a signal for intrusion; otherwise, no signal is produced from the information fusion technique for the overall assessment of intrusion likelihood.

More details of the individual intrusion detection techniques and the information fusion technique can be found in (Ye et al. 1, 2000; Ye et al. 2, 2000; Ye & Li, 2000; Ye et al. 3, 2000; Ye, 2000).

In testing, the Event Dispatcher of the Centralized IDS Server scans through the data only once. For each object associated with an event an Individual Technique Servers calculate observation values for the variables of the object. From the observation values an Individual Technique Server calculates the *T* value and then IW level ($0 \leq IW \leq 1$). The IW values calculated by the Individual Technique Servers are fed into fusion to get the composite IW level ($IW_{composite}$: 0 for normal, 1 for intrusive), which is used to label the testing data into normal or intrusive. The training process is complex and requires several passes over the input event stream. Training should be done periodically when requested or set by SSA.

Figure 1-6 presents the performance of the X2 technique based on the Receiver Operating Characteristic (ROC) analysis of session signal ratio. The CANB technique is similar to the X2 technique. Hence, it is not tested. Figure 1-7 presents the performance of the CL technique based on the Receiver Operating Characteristic (ROC) analysis of session signal ratio. The testing of information fusion is still ongoing, and thereby is not presented here.

ROC analyzes the tradeoff between false alarm and hit rates for detection systems. A false alarm is a signal on an event when in fact the event is normal. A false alarm rate is ratio of the number of false alarms to the number of all the normal events in the testing data. A hit is a signal on an event when in fact the event is intrusive. A hit rate is the ratio of the number of hits to the number of all the intrusive events in the testing data. ROC was developed in the field of signal detection (Egan, 1975; Swets, 1973). It has now become the standard approach to evaluate detection systems. ROC curves for intrusion detection indicate how the hit rate changes as the signal threshold varies to generate more or fewer false alarms to tradeoff detection accuracy against analyst workload. Measuring the hit rate alone indicates only intrusions that an intrusion detection system may detect, and it does not indicate the human workload required to analyze false alarms generated by the normal background traffic. A low false alarm rate with a high hit rate means that the detection output can be trusted and human labor required to confirm any detection is minimized. For a given signal threshold, a pair of a hit rate and a false alarm rate can be computed. Hence, by varying the signal threshold, a curve (ROC curve) can be plotted. The closer the ROC curve is to the top-left corner representing the 100% hit rate and the 0% false alarm rate, the better the detection performance is.

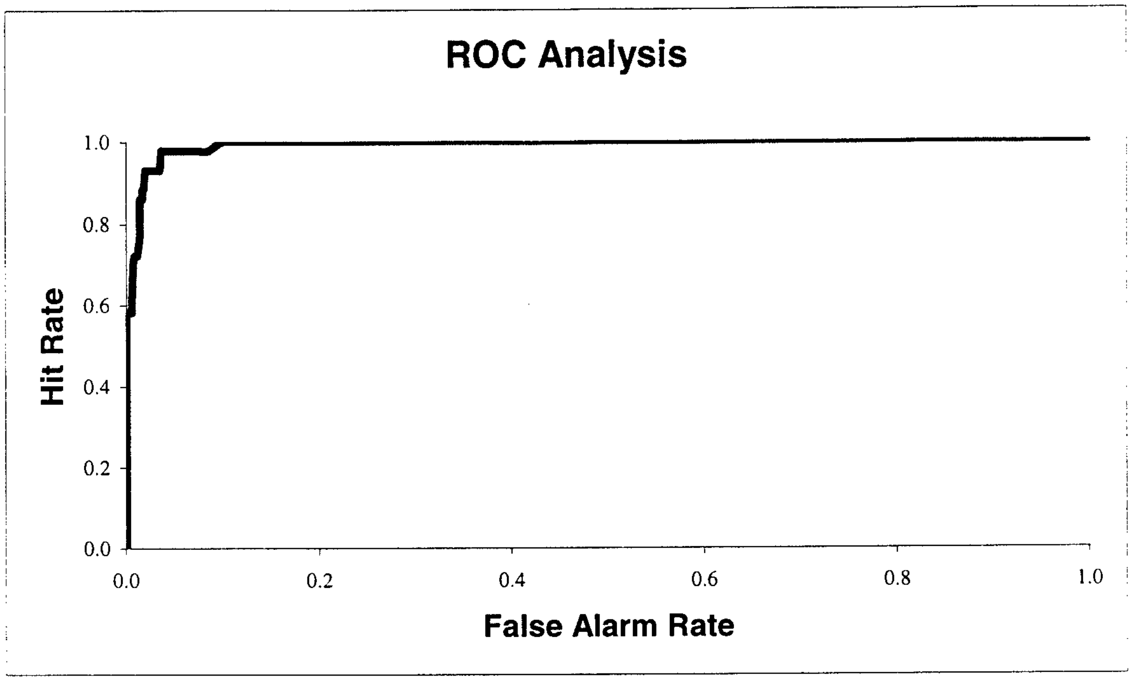


Figure 1-6. The performance of the X2 technique.

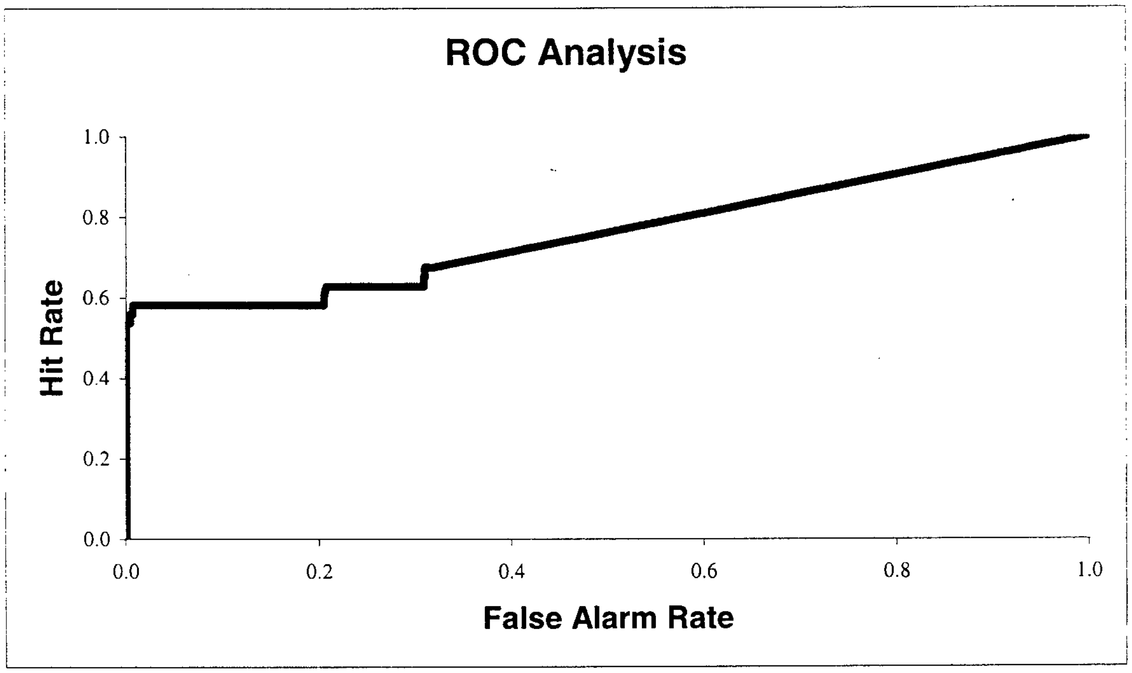


Figure 1-7. The performance of the CL technique.

Since intrusive events occur in sessions, we analyze the results in terms of session signal ratio. We group the events according to sessions and count how many of the events inside a session are signaled and we call the ratio of number of signals to number of events in a session as session signal ratio. If it is an intrusive session then we expect a high session signal ratio. If it is a normal session, we expect the session signal ratio to be low. In this paper we present the ROC curves for the X2 technique and the CL technique using the session signal ratio as shown in Figures 1-6 and 1-7.

The X2 technique achieves the 60% hit rate at the 0% false alarm rate. At the 2% false alarm rate the hit rate of the X2 technique rises to 85%. The technique achieves the 100% hit rate after the 10% false alarm rate. Therefore, the X2 technique produces good detection performance.

When there is no false alarm produced by the CL technique, the hit rate reaches 53.5%. If we want to limit the false alarm rate to no more than 10%, we get the 58.1% hit rate.

Hence, the X2 anomaly detection technique has better intrusion detection performance than the CL signature recognition technique. This performance difference is expected because a signature recognition technique can detect only known intrusions used in the training, whereas an anomaly detection technique can detect both known and novel attacks if they demonstrate significant deviations from normal behavior.

The report in (Lippmann et al., 2000) presents the performance of some other IDSs evaluated. The performance evaluation of those IDSs is based on the whole set of the 1998 DARPA-LL training and testing data, whereas our training and testing of ISA-IDS are based on only four-day data. Hence, we cannot draw conclusions with regard to the comparison of those IDSs and ISA-IDS in their intrusion detection performance. If we simply examine the ROC curves of those IDSs and ISA-IDS, the X2 anomaly detection technique performs much better than the anomaly detection techniques in those IDSs in (Lippmann et al., 2000). The CL signature recognition does not perform as well as the best signature recognition technique in those IDSs in (Lippmann et al., 2000). However, all the signature recognition techniques in those IDSs in (Lippmann et al., 2000) uses some form of manual tuning (e.g., the manual input of rules, and the manual selection of features) to achieve better performance, whereas the training of our CL signature recognition technique was done automatically without any manual tuning.

VI. SUMMARY

In this chapter we have presented the architecture and implementation details of ISA-IDS that is tested using the 1998 DARPA-LL audit data. During training it calculates the training parameters for each of the techniques. During testing, for each audit event on each object ISA-IDS applies a selective set of four intrusion detection techniques to find out the level of intrusion and fuses them into a single IW value, 0 for normal and 1 for intrusive. In ISA-IDS, we incorporate both a signature recognition technique and several anomaly detection techniques along with an information fusion technique. The techniques in ISA-IDS are scalable to large amounts of audit data, where many existing signature recognition techniques and anomaly detection techniques cannot scale to large audit data in real time.

The ISA-IDS prototype currently uses the 1998 DARPA-LL BSM audit data for testing. The data come from one single host machine. We can incorporate data from multiple host machines as well. The Event Data Collector has to be developed and installed in each of the hosts. The system currently uses off-line audit data. The testing can be run in real time, and the training can be run periodically.

REFERENCES

Axelsson, S. (1998). Research in Intrusion-Detection systems: A Survey. *Technical Report 98-17.* Dept. of Computer Engineering, Chalmers University of Technology, Goteborg, Sweden.

Balasubramaniyan, J., Garcia-Fernandez, J.O., Isacoff, D., Spafford, E.H., & Zamboni, D. (1998). An Architecture for Intrusion Detection using Autonomous Agents. *Technical Report 98-05.* Center for Education and Research in Information Assurance and Security, Department of Computer Sciences, Purdue University.

Chen, S.S., Cheung, S., Crawford, R., Dilger, M., Frank, J., Hoagland, J., Levitt, K., Wee, C., Yip, R., & Zerkle, D. (1996). GrIDS: A graph based intrusion detection system for large networks. *Proceedings of the 19th National Information Systems Security Conference*, volume 1, pp. 361-370. National Institute of Standards and Technology.

Debar, H., Becker, M., & Siboni, D. (1992). A neural network component for an intrusion detection system. *Proceedings of the 1992 IEEE Computer Society Symposium on Research in Security and Privacy*, pp. 240-250.

Debar, H., Dacier, M., & Wespi, A. (1999). Towards a taxonomy of intrusiondetection systems. *Computer Networks*, *31*(8), pp. 805-822.

Durst, R., Champion, T., Witten, B., Miller, E., & Spagnuolo, L. (1999). Testing and evaluating computer intrusion detection systems. *Communications of the ACM*, *42*(7), pp. 53-61.

Egan, J.P. (1975). *Signal detection theory and ROC-analysis.* New York: Academic Press.

Forrest, S., Hofmeyr, S.A., & Somayaji, A. (1997). Computer Immunology. *Communications of the ACM*, *40*(10), pp. 88-96.

Hochberg, J., Jackson, K., Stallings, C., McClary, J.F., DuBois, D., & Ford, J. (1993). NADIR: An automated system for detecting network intrusion and misuse. *Computers and Security*, *12*(3), pp. 235-248.

Ilgun, K. (1993). USTAT: A real-time intrusion detection system for UNIX. *Proceedings of the 1993 IEEE Symposium on Security and Privacy*, pp. 16-28.

Jagannathan, R., Lunt, T., Anderson, D., Dodd, C., Gilham, F., Jalali, C., Javitz, H., Neumann, P., Tamaru, A., & Valdes, A. (1998). System Design Document: Next-Generation Intrusion Detection Expert System (NIDES). *Technical Report.* Computer Science Laboratory, SRI International, 1998.

Kendall, K. (1999). A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems. *M.S. Thesis.* MIT Department of Electrical Engineering and Computer Science.

Ko, C., Ruschitzka, M., & Levitt, K. (1997). Execution monitoring of security-critical programs in distributed systems: A specification-based approach. *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pp. 175-187.

Kumar, S., & Spafford, E.H. (1995). A software architecture to support misuse intrusion detection. *Technical report.* The COAST Project, Department of Computer Sciences, Purdue University.

Lee, W., Stolfo, S.J., & Mok, K. (1999). A Data Mining Framework for Building Intrusion Detection Models. *Proceedings of IEEE Symposium on Security and Privacy.*

Lippmann, R.P., Fried, D.J., Graf, I., Haines, J.W., Kendall, K.R., McClung, D., Weber, D., Webster, S.E., Wyschogrod, D., Cunningham, R.K., & Zissman, M.A. (2000). Evaluating Intrusion Detection Systems: The 1998 DARPA Off-line Intrusion Detection Evaluation.

Proceedings of DARPA Information Survivability Conference and Exposition (DISCEX) 2000, volume 2, pp. 12-26.

Lunt, T.F., Tamaru, A., Gilham F., Jagannathan, R., Jalali, C., & Neuman, P.G. (1992). A real-time intrusion-detection expert system (IDES). *Technical Report Project 6784*. Computer Science Laboratory, SRI International.

Northcutt, S. (1999). *Network Intrusion Detection: An Analyst's Handbook*. New Riders Publishing.

Porras, P.A., & Neumann, P.G. (1997). EMERALD: Event monitoring enabling responses to anomalous live disturbances. *Proceedings of the 20th National Information Systems Security Conference*. National Institute of Standards and Technology.

Ryan, T.P. (2000). *Statistical Methods for Quality Improvement*. John Wiley & Sons.

Smaha, S. (1988). Haystack: An intrusion detection system. *Proceedings of the IEEE Forrth Aerospace Computer Security Applications Conference.*

Swets, J.A. (1973). The Relative Operating Characteristic in Psychology. *Science*, *182*, pp. 990–1000.

White, G.B., Fisch, E.A., & Pooch, U.W. (1996). Cooperating security managers: A peer-based Intrusion Detection System. *IEEE Network*, pp. 20-23.

Ye, N. (2000). A Markov Chain Model of Temporal Behavior for Anomaly Detection. *Proceedings of IEEE Systems, Man and Cybernetics Information Assurance & Security Workshop*. United States Military Academy.

Ye, N., Chen, Q., Emran, S.M., & Noh, K. (2000). Chi-square Statistical Profiling for Anomaly Detection. *Proceedings of IEEE Systems, Man and Cybernetics Information Assurance & Security Workshop*. United States Military Academy.

Ye, N., Chen, Q., Emran, S.M., & Vilbert, S. (2000). Hotelling's T^2 Multivariate Profiling for Anomaly Detection. *Proceedings of IEEE Systems, Man and Cybernetics Information Assurance & Security Workshop*. United States Military Academy.

Ye, N., & Li, X. (2000). Application of Decision Tree Classifier to Intrusion Detection. *Proceedings of Second International Conference on DATA MINING 2000*. Cambridge University.

Ye, N., Li, X., & Emran, S.M. (2000). Decision Tree for Signature Recognition and State Classification. *Proceedings of IEEE Systems, Man and Cybernetics Information Assurance & Security Workshop*. United States Military Academy.

Ye, N., & Xu, M. (2000). A Process Model Based Integration of Intrusion Detection Techniques. *Proceedings of Industrial Engineering Research Conference*.

Chapter 2: Attributes of System Activity Data for Intrusion Detection

This chapter presents various attributes of system activity data for intrusion detection. These attributes are tested to examine their effectiveness for intrusion detection.

I. INTRODUCTION

Vulnerabilities and bugs of information systems are often exploited by malicious users to intrude into information systems and compromise security (e.g., availability, integrity and confidentiality) of information systems [2-1 – 2-10]. As information systems become increasingly complex, vulnerabilities and bugs of information systems are inevitable for technical and economic reasons. Hence, the possibility of intrusions into information systems always exists. In order to protect information systems, it is highly desirable to detect intrusive activities while they are occurring in information systems.

An information system consists of host machines and communication links between host machines. Existing intrusion detection efforts [2-11 – 2-43] focus mainly on two sources of activity data in an information system: network traffic data and computer audit data. Network traffic data contain data packets traveling over communication links between host machines, and thus capture activities over communication networks. Audit trail data capture activities occurring on individual host machines. Activity data of an information system contain not only useful information to uncover intrusive activities but also much irrelevant information.

This chapter presents a series of studies performed at the Information and Systems Assurance Laboratory of Arizona State University to reveal a few probabilistic properties of computer audit data that are important to intrusion detection. Intrusion detection techniques, including decision tree, Hotelling's T^2 test, chi-square multivariate test and Markov chain, are used in these studies. Section II review attributes of activity data used in existing work on intrusion detection. Section III generalizes probabilistic properties from attributes of activity data. Sections IV describes computer audit data used in our studies, and presents these studies and their results concerning probabilistic properties of activity data. Section V gives a conclusion.

II. ATTRIBUTES OF ACTIVITY DATA IN EXISTING WORK

There are two general approaches to detecting intrusions [2-11 – 2-43]: anomaly detection (named behavior-based approach in some literature [2-11]) and pattern recognition (named knowledge-based approach [2-11] or misuse detection [2-29] in some literature). Pattern recognition techniques [2-11, 2-16 – 2-25] identify and store signature patterns of known intrusions, match activities in an information system with known patterns of intrusion signatures, and signal intrusions when there is a match. Pattern recognition techniques are efficient and accurate in detecting known intrusions, but cannot detect novel intrusions whose signature patterns are unknown.

Anomaly detection techniques establish a profile of a subject's normal activities (a norm profile), compare observed activities of the subject with its norm profile, and signal intrusions when the subject's observed activities differ largely from its norm profile [2-26 – 2-43]. The subject may be a user, file, privileged program, host machine, or network. Denning [2-29] provides a justification of the anomaly detection approach to intrusion detection. Anomaly detection techniques can detect both novel and known attacks if they demonstrate large differences from the norm profile. Since anomaly detection techniques signal all anomalies as intrusions, false alarms are expected when anomalies are caused by behavioral irregularity instead of intrusions. Hence, pattern recognition techniques and anomaly detection techniques are often used together to complement each other.

Existing efforts on intrusion detection have considered mainly the following attributes of activities in information systems:

1) Occurrence of individual events, e.g., audit events, system calls, commands, error messages, IP source address, and so on,
2) Frequency of individual events, e.g., number of consecutive password failures,
3) Duration of individual events, e.g., CPU time of a command, and duration of a connection,
4) Occurrence of multiple events combined through logical operators such as AND, OR and NOT,
5) Frequency histogram (distribution) of multiple events, and
6) Sequence or transition of events.

Attributes 1, 2, 4 and 6 often appear in intrusion signatures that are represented in manually coded rules [2-16 – 2-18] or automatically learned rules [2-19 – 2-22] in some pattern recognition techniques. Attribute 6 appears in state transition diagrams [2-23 – 2-24] and colored Petri nets [2-25] that are used in some pattern recognition techniques to represent intrusion signatures.

Several anomaly detection techniques exist and differ in the representation of a norm profile and the inference of a deviation from the norm profile. Specification-based anomaly detection techniques describe security policies and authorized activities of a well-defined subject (e.g., a privileged program or a network server) in terms of formal logic and activity graph [2-26 – 2-28]. Statistical-based anomaly detection techniques build a statistical profile (e.g., statistical distribution) of a subject's normal activities from historic data [2-29 – 2-33]. Anomaly detection techniques based on regression [2-34] or artificial neural networks [2-35 – 2-36] learn from historic data to predict the next event from a series of the past events. Anomaly detection techniques based on immunology capture a large set of event sequences as the norm profile from historic data of a subject's normal activities, and use either negative selection or positive selection algorithms to detect the difference of incoming event sequences from event sequences in the norm profile [2-36 – 2-39]. There are also anomaly detection techniques that use a first-order or high-order Markov model of event transitions to represent a norm profile [2-39 – 2-43]. A first-order model of event transitions assumes that the next event depends on only the last event in the past. A higher-order Markov model of event transitions assumes that the next event depends on multiple events in the past.

Specification-based anomaly detection techniques can readily incorporate attributes 1, 2, 4 and 6. Statistical-based anomaly detection techniques can build a statistical norm profile based on attributes 2, 3 and 5. Attribute 6 is incorporated in anomaly detection techniques based on regression, artificial neural networks, immunology, and Markov models.

Although attributes 1-6 appear in existing work on intrusion detection, it is not clear which properties are critical to performance of intrusion detection for several reasons. First, existing studies investigating different attributes often use different data sets, making the comparison of activity data attributes difficult. Second, existing comparative studies using the same data set focus mainly on differences among various intrusion detection algorithms dealing with the same attribute of activity data [2-39].

The following sections present generalized probabilistic properties of activity data and our comparative studies on these probabilistic properties concerning their importance to intrusion detection.

III. PROBABILISTIC PROPERTIES OF ACTIVITY DATA

Attributes 1-6 can be categorized into three groups: attributes 1, 2, 4 and 5 concerning the frequency property of events, attribute 3 concerning the duration property of events, and attribute 6 concerning the ordering property of events. There may be other aspects of activity data that are not represented by these three properties. This paper focuses on only these three properties.

For the frequency property of events, we can use a set of random variables, $(X_1, \ldots, X_n)$, to represent the frequency of n different types of events (e.g., commands, system calls or audit events) for a given sequence of events. If we are interested in attributes 1 and 2 – single or multiple occurrences of the ith event type from a pool of n possible event types for a given sequence of events, we examine the value of X_i from the vector of $(X_1, \ldots, X_n)$. A denial-of-service attack may manifest through an unusually high frequency of a single event type. If we are interested in attributes 4 and 5 – single or multiple occurrences of multiple events in combination, we examine the multivariate frequency distribution of $(X_1, \ldots, X_n)$.

For the duration property of events, we can use n sets, $\{X_1\}, \ldots, \{X_n\}$, to represent the duration values of n different event types for a given sequence of events, where each set contains duration values of events of a certain type. Considering the execution of a program as an event, the duration of this event is the program execution time. A Trojan Horse program may manifest through a change in the program execution time.

For the ordering property of events, (X_1, ..., X_t) gives a time-series representation of a given event sequence, where X_t denotes an event occurring at time *t*. To take into account the ordering of events, complex data models are usually required, as demonstrated by a number of studies [2-34 – 2-43]. These complex data models demand for computationally intensive learning and/or inference procedures that are not scalable to mountains of activity data from an information system. An information system, even as small as a host machine, usually produces large amounts of activity data at a high frequency regardless of whether there are users' application processes in the information system. The large computational overhead associated with the ordering property raises a question: is the ordering property of events necessary for intrusion detection?

Note that the vector representation of the frequency property can be derived from the set representation of the duration property and from the times-series representation of the ordering property. An intrusion typically consists of a series of events in an information system. The vector representation of the frequency property contains the least amount of information about the given series of events among the representations of the three properties. However, the frequency distribution of multiple event types still provides information about the collective activity level of these event types. To answer the question on the order property, we can answer another question: is the frequency property sufficient for intrusion detection? If the frequency property of multiple event types for a given sequence of events is sufficient to produce good intrusion detection performance, another question follows: can intrusion detection be stateless, in other words, is a single event at a given time sufficient to detect intrusions? Section IV presents a series of studies to answer these questions.

IV. COMPARATIVE STUDIES

Our studies use various probabilistic techniques to detect intrusions, including decision tree, Hotelling's T2 test, chi-square multivariate test and Markov chain, because different properties of activity data require different data models. Decision tree is a data mining technique that we use here as a pattern recognition technique. The other techniques are anomaly detection techniques. Before we present these studies, we first describe computer audit data used in these studies.

A. Training and Testing Data

An anomaly detection technique requires data of only normal activities in an information system to build a norm profile. Data of both normal activities and intrusive activities in an information system are required to learn intrusion signatures for a pattern recognition technique. The testing data should contain data of both normal activities and intrusive activities to test the performance of intrusion detection.

We use computer audit data from a Sun SPARC workstation with the Solaris operating system, and focus on intrusions into a host machine that leave trails in computer audit data. The Solaris operating system from the Sun Microsystems Inc. has a security extension, called the Basic Security Module (BSM). BSM supports the monitoring of activities on a host machine by recording security-relevant events. BSM auditable events fall into two categories: kernel events and user-level events. Kernel events are generated by system calls to the kernel of the Solaris operation system. User-level events are generated by application software.

There are more than 250 different types of BSM auditable events, depending on the version of the Solaris operating system. Since there are about 284 different types of BSM audit events on our host machine, we consider 284 event types in our studies. An BSM audit record for each event contains a variety of information, including the event type, user ID, group ID, process ID, session ID, the system object accessed, etc. In our studies, we extract and use only the event type, because many existing studies [26-43] use only the type of events and produce good intrusion detection performance. Hence, activities on a host machine are captured through a stream of audit events, each of which is characterized by the event type.

A sample of computer audit data of normal activities is downloaded from the MIT (Massachusetts Institute of Technology) Lincoln Lab at http://ideval.ll.mit.edu/1998/1998_index.html. At this web site, four sets of data are provided: sample data, four-hour subset of training data, seven weeks of training data, and two weeks of testing data. We download the sample data which contain audit data of both normal and intrusive activities. These audit data of normal activities are generated by the MIT Lincoln Lab through simulating activities in a real information system used by the US Air Force. Intrusions are simulated on the background of normal activities. Because intrusive activities in this small sample data are very limited, we use audit data of normal activities only from this sample data.

According to the provided description of the starting and ending times of attack activities, we obtain a block of audit data for the period when no attack activities occur. This block of audit data containing a stream of 3019 audit events [2-12]. These audit data of normal activities is divided into two different parts: the first 1613 audit events and the remaining 1406 audit events.

The first part, consisting of 1613 audit events, is included in the training data set as the computer audit data of normal activities. The second part, consisting of 1406 audit events, is included in the testing data set as the computer audit data for normal activities.

Computer audit data of intrusive activities are generated in our laboratory by simulating 15 intrusion scenarios that we have collected over years from various information sources. Table 2-1 gives the description of these intrusion scenarios. These intrusion scenarios are simulated in a random order. A student manually runs these intrusion scenarios on a Sun SPARC workstation with the same version of the Solaris operating system as the Solaris operating system that is used to generate audit data at the MIT Lincoln Laboratory, while the auditing facility is turned on. These intrusion scenarios generate a stream of 1751 audit events. The first 526 audit events produced from the first 8 intrusion scenarios are included in the training data set as the computer audit data of intrusive activities. The remaining 1225 audit events from the remaining 7 intrusion scenarios are included in the testing data set as the computer audit data of intrusive activities.

Hence, the training data set contains 1613 audit events of normal activities and 526 audit events of intrusive activities. The testing data contains 1406 audit events of normal activities and 1225 audit events of intrusive activities. Although audit data of normal activities and audit data of intrusive activities come from different host machines, the same Solaris operating system on these host machines produces the consistent information about event types. Since only the event type is extracted from each audit record, putting together audit data from different host machines in the training data set and the testing set does not become a concern in this study.

Table 2-1. Description of intrusion scenarios used in the study.

Scenario	**Description of intrusive activities**

Number	
1	Link the printer driver to a user program, and then execute the printing program. Before the printing program completes the job, replace the link with reference to the user program, in order to bypass the security check and let the user program obtain the same priority of the printing program.
2	Use the command of *rcp* to copy the password file from a remote host
3	Link the password file with the dead.letter. The dead.letter is created by the system when there are dead (return) mails The system has the "write" right on the password file, which makes it possible for the user to change the password file through the dead mails.
4	Edit the *.rhost* file in order to remotely access the host later.
5	Attempt to edit and view the password file.
6	Execute a binary executable code that needs /dev/ttyb as an argument.
7	Attempt to login, but fail three times
8	Link the password file with a temporal log file that system could generate, in order to overwrite the password file later.
9	Use the command of *lp* to print a linked long file. Before the printing job is done, replace the link with reference to the password file, in order to print the password file.
10	Use the command of *rlogin* to login on a remote host without a password, provided that the account is in the *.login* file and the on the .rhosts file on the host.
11	Use the command of *finger* to get sensitive information from a remote host.
12	Link a user file to a system-generated file.
13	Move system files for creating an executable file under the system directory to make a regular user become the root user.
14	Use the command of *rsh* to view and edit the password file in a remote server.
15	Link the shadow password file to a temporal log file that system could generate, in order to overwrite the password file later.

The same set of training data and the same set of testing data are used by each intrusion detection technique in our studies. Although the training data set contains computer audit data of both normal activities and intrusive activities, an anomaly detection technique uses only computer audit data of normal activities to build a norm profile. A pattern recognition technique uses computer audit data of both normal activities and intrusive activities in the training data set to learn intrusion signatures. Each intrusion detection technique is tested using the entire set of the testing data, containing computer audit data of both normal activities and intrusive activities. The training and the testing of each intrusion detection technique are performed off-line using one file of the training data set and another file of the testing data set respectively.

B. Decision Tree and Results

There are two kinds of variables in a decision tree. One is target variable or class, and the other is predictor variable. A data point in the training data set contains the values of both predictor variables and a target variable. That is, the training data are labeled by the target values. A data point in the testing data set contains the values of only predictor variables. After testing, each data point in the testing data is assigned a target value and classified by the target value.

During training, a decision tree is constructed by recursively partitioning data points in the training data set into branches according to values of predictor variables until a stopping criterion is met [2-44 – 2-47]. Each branch contains a subset of data points with less inconsistency with respect to their target values. A common stopping criterion for a branch is that all data points in the branch have the same target value, which then produces a leaf in the decision tree. During testing, a data point is passed through the decision tree to reach a leaf according to its values of predictor variables, and is assigned the target value of this leaf.

Decision tree is used as a pattern recognition technique in our studies to construct a decision tree from the training data and to classify the testing data. Each data point in the training data set is labeled by the value of the target variable to indicate whether it is normal or intrusive. A normal data point has 0 as the target value. An intrusive data point has 1 as the target value. After training, each path from the root to a leaf of the decision tree represents a pattern of activities. We assign an Indications and Warning (IW) value to each leaf in the decision tree to indicate the likelihood of intrusion. The IW value for a leaf is computed by averaging the target values of the training examples in that leaf. The higher the IW value, the more likely the leaf represents intrusive activities. During testing, a data point is classified into a leaf according to the values of its predictor variables, and takes the IW value of the leaf.

We use the CHAID (Chi-squared Automatic Interaction Detector) decision tree algorithm in the AnswerTree 2.0 software from SPSS [2-48] for learning a decision tree from our training data. The CHAID algorithm uses chi-square statistics to identify optimal partitions. Details of our implementation can be found in [2-49].

To answer the question on whether a single event is sufficient to detect intrusions, we develop two different representations of predictor variables: a single-event representation and a frequency distribution representation. The single-event representation considers only a single event at a given time. The frequency distribution representation considers the frequencies of multiple event types within a given sequence of events.

In the single-event representation, there is only one predictor variable, X, with a value corresponding to the event type at a given time. Since there are 284 possible event types, the predictor variable can take one of 284 possible values.

In the frequency distribution representation, there are 284 predictor variables, (X_1, X_2, ..., X_{284}), for 284 event types respectively. The value for each of 284 predictor variables represents the frequency of an event type within a given sequence of audit events. We use the exponentially weighted moving average method [2-50] to compute the value of X_i, that is, the frequency of the ith event type in a sequence of audit events in the recent past.

$X_i(t) = \lambda * 1 + (1-\lambda) * X_i(t-1)$ if the current event – event t - belongs to the ith event type (4)

$X_i(t) = \lambda * 0 + (1-\lambda) * X_i(t-1)$ if the current event - event t - is different from the ith event type where $X_i(t)$ is the observed value of the i^{th} variable in the vector of an observation (X_1, X_2, ..., X_{284}) for the current event - event *t*, λ is a smoothing constant that determines the decay rate, and $i = 1, \ldots, 284$.

By using the exponentially weighted moving average method, more recent observations receive larger weights in the frequency computation. For example, the observation at the current event – event t - receives a weight of λ, the $(t-1)^{th}$ observation receives a weight *of* $\lambda(1-\lambda)$, and the $(t-k)^{th}$ observation receives a weight of $\lambda(1-\lambda)^k$. An observation is made at each event.

In our studies, we let λ be 0.3 - a commonly used value for the smoothing constant [2-48]. Figure 2-1 shows the decay effect of the smoothing constant 0.3. We can see from Figure 2-1 that after the $(t-14)^{th}$ observation (k=14) the weight drops close to zero. That is, the frequency value of $X_i(t)$ at the current event – event *t* - takes into account about the past 15 audit events (k = 0, ..., 14). We initialize $X_i(0)$ to 0 for i = 1, ..., 284.

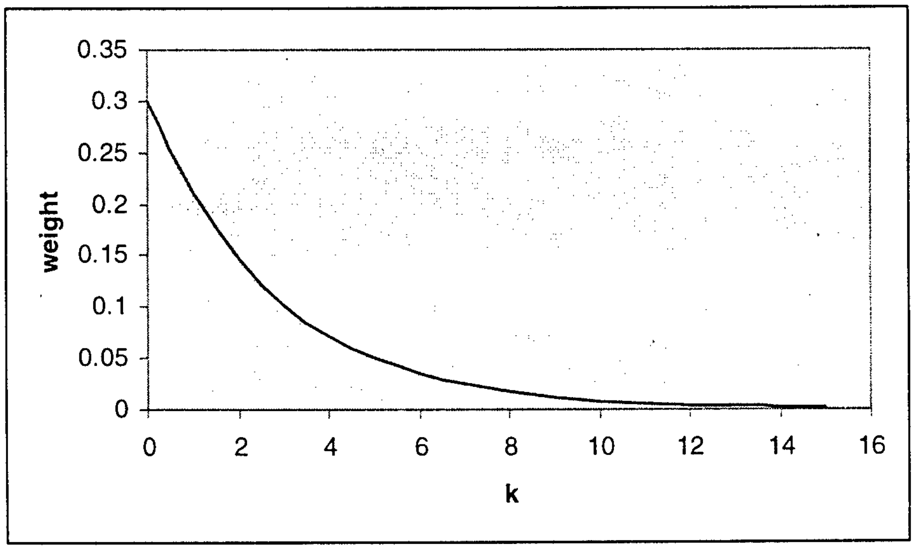


Figure 2-1. The decay effect of the smoothing constant 0.3.

If we take a real time unit (e.g., second) for t in $X_i(t)$, the frequency distribution representation can convey not only the relative frequency distribution of 284 audit events in a stream of audit events for a given timeframe, but also the intensity of individual events for that timeframe. However, the intensity of activities in an information system has large variations over time, e.g., from day to night. At night, there may be little activities in the information system. Inactive periods do not give us accurate estimates of the relative frequency distribution of audit events. Hence, we have separate studies on the intensity of individual events and the relative frequency distribution of multiple events. Another paper reports our intrusion detection work that examines the intensity of individual events for a given timeframe [2-51]. This paper focuses on the relative frequency distribution of 284 audit events within a given sequence of events by making an observation at every event rather than every time unit.

To accurately capture the relative frequency distribution of 284 audit events, we number only time points when audit events are observed in the frequency distribution representation. For example, given the following stream of audit events, we number them 1, 2, 3, ... for time t:

```
t = 0,  1,            2,            3,            ......
        EventType3,   EventType8,   EventType1,   ......
```

For each audit event in the training data and the testing data, we obtain an observation vector of (X_1, ..., X_{284}). For the above example, at t = 0, all variables in the vector of (X_1, ..., X_{284}) have a value of 0. At time t = 1, X_3 has a value of 0.3 (=0.3*1+0.7*0), and all other variables have a value of 0. At time t = 2, X_3 has a value of 0.21 (=0.3*0+0.7*0.3), X_8 has a value of 0.3 (=0.3*1+0.7*0), and all other variables have a value of 0. At t = 3, X_3 has a value of 0.147 (=0.3*0+0.7*0.21), X_8 has a value of 0.21 (=0.3*0+0.7*0.3), X_1 has a value of 0.3 (=0.3*1+0.7*0), and all other variables have a value of 0.

Using the single-event representation, we obtain 1613 data points for normal audit events and 526 data points for intrusive audit events in the training data set. Each data point contains the value of a predictor variable, X, and a target value. Using such training data, the CHAID algorithm produces a decision tree, called the single-event decision tree (SEDT). SEDT is then used to obtain an IW value for each of 1406 data points for normal audit events and 1225 data points for intrusive data events in the testing data set. Figure 2-2 shows the ROC (Receiver Operator Characteristic) curve of the SEDT testing results.

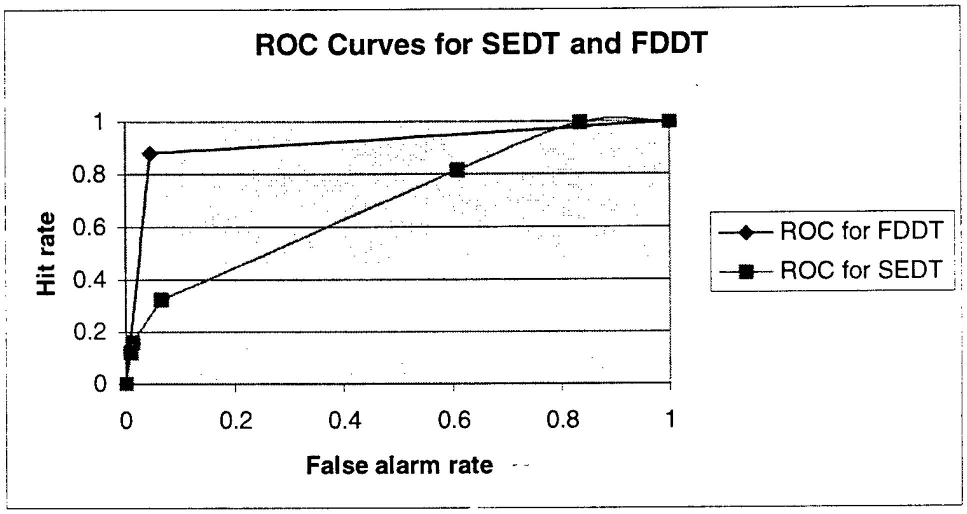


Figure 2-2. ROC curves of decision trees.

Each point in an ROC curve indicates a pair of the hit rate and the false alarm rate for a signal threshold. For example, if the signal threshold for the IW values of the testing data is set to 0.5, we signal a testing data point whose IW value is greater than or equal to 0.5 as intrusive. There are no signals on testing data points whose IW values are less than 0.5. If there is a signal on a data point for an intrusive event in the testing data, this is a hit. If there is a signal on a data point for a normal event in the testing data, this is a false alarm. The hit rate is computed from dividing the total number of hits by the total number of intrusive events in the testing data. The false alarm rate is computed from dividing the total number of false alarms by the total number of normal events in the testing data. By varying the value of the signal threshold, we obtain an ROC curve. The closer the ROC is to the top-left corner (representing 100% hit rate and 0% false alarm rate) of the chart, the better detection performance the intrusion detection technique yields.

Using the frequency distribution representation, we also obtain 1613 data points for normal audit events and 526 data points for intrusive audit events in the training data set. Each data point contains an observation vector (X_1, X_2, ..., X_{284}) and a target value. Using such training data, the CHAID algorithm produces a decision tree, called the frequency-distribution decision tree (FDDT). FDDT is then used to obtain an IW value for each of 1406 data points for normal audit events and 1225 data points for intrusive data events in the testing data set. The ROC curve of the FDDT testing results is shown in Figure 2-2.

The ROC curves from the SEDT testing results and the FDDT testing results reveal much better intrusion detection performance of FDDT than that of SEDT. In fact, the intrusion detection performance of SEDT is poor. For SEDT a hit rate of 81.5% brings up the false alarm rate to 60.9%, whereas for FDDT a hit rate of 88.1% brings up the false alarm rate to only 4.6%. Hence, the relative frequency of multiple event types within a given sequence of events gives a great advantage to intrusion detection. In other words, the frequency property of activity data is necessary for intrusion detection. Since a single event is not sufficient to produce good intrusion detection performance, stateless intrusion detection is not recommended.

Decision tree is a pattern recognition technique in which the learning of intrusion signatures requires both normal audit data and intrusive audit data. The importance of the frequency property of activity data to intrusion detection is further verified below through two anomaly detection techniques (Hotelling's T2 test and chi-square multivariate test) that use only normal audit data for training.

C. Hotelling's T^2 Test, Chi-square Multivariate Test and Results

Hotelling's T^2 test is a multivariate statistical process control technique that detects anomalies in a process of a system. Let $\mathbf{X} = (X_1, X_2, \ldots, X_p)$ denote an observation of *p* variables from a process at time *t*. Using a data sample of size *n*, the sample mean vector $\overline{\mathbf{X}}$ and the sample variance-covariance matrix *S* of *p* variables are determined as follows [2-52]:

$$\overline{\mathbf{X}} = (\overline{X_1}, \overline{X_2}, \ldots, \overline{X_p}) \tag{2-1}$$

$$\mathbf{S} = \frac{1}{n-1} \sum_{i=1}^{n} (\mathbf{Xi} - \overline{\mathbf{X}})(\mathbf{Xi} - \overline{\mathbf{X}})'. \tag{2-2}$$

Hotelling's T^2 statistic for an observation, $\mathbf{X}$, is determined as follows [2-52]:

$$T^2 = (\mathbf{X} - \overline{\mathbf{X}})' S^{-1} (\mathbf{X} - \overline{\mathbf{X}}). \tag{2-3}$$

A large computed value of T^2 indicates a large deviation of the observation $\mathbf{X}$ from the in-control population. Details of Hotelling's T^2 test and its application to intrusion detection can be found in [2-53].

When we apply Hotelling's T^2 test to intrusion detection, we use the same training data and the same testing data as those in decision tree studies, except that only audit events of normal activities are used for training a norm profile for Hotelling's T^2 test. Since only 11 event types actually appear in the training data set of 1613 audit events, the vector X of $(X_1, X_2, \ldots, X_{284})$ is reduced into a vector $\mathbf{X}$ with only eleven variables for the eleven event types respectively. We perform the training and the testing for Hotelling's T^2 test using the vector $\mathbf{X}$ with only 11 variables. That is, using 1613 data points of **X** with 11 variables for normal audit events in the training date set, we compute $\overline{\mathbf{X}}$ and S in formulas (2-1) and (2-2) which fully describe the norm profile.

Using $\overline{\mathbf{X}}$ and S, the T^2 value in formula (2-3) is then computed for each data point in the testing data set. The computed T^2 value is small if the data point conforms to the norm profile. The ROC curve for the testing results of Hotelling's T^2 test is plotted using various signal thresholds on the values of the computed T^2 value for the testing data points, as shown in Figure 2-3.

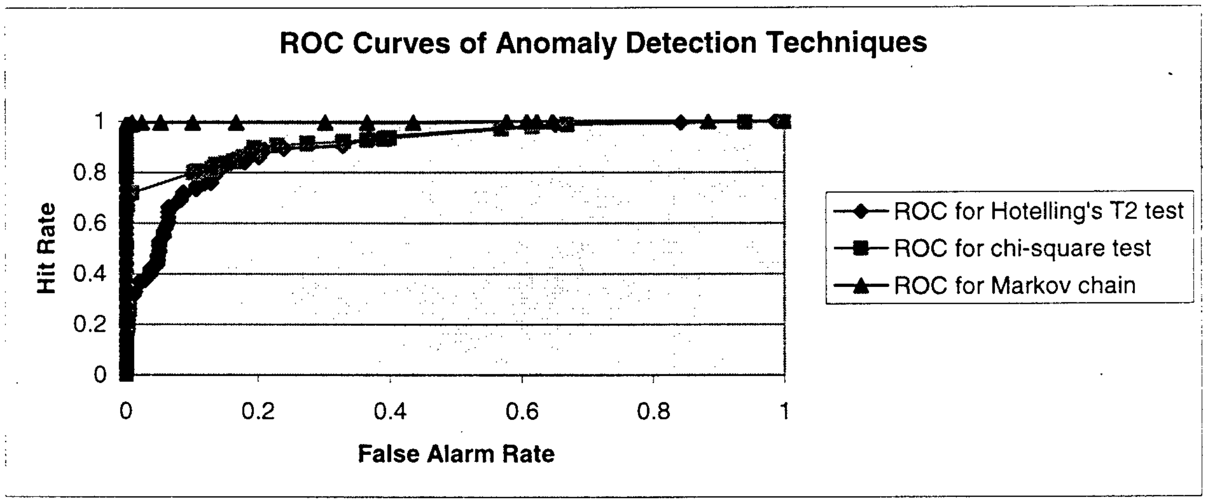


Figure 2-3. ROC curves of anomaly detection techniques.

With both the mean vector $\overline{\mathbf{X}}$ and the variance-covariance matrix S, Hotelling's T^2 test provides a complete data model of multivariate data **X** in the frequency-distribution representation of the frequency property. Hotelling's T^2 test detects both mean shifts and counter-relationships in a multivariate manner. However, Hotelling's T^2 test is computationally intensive, requiring large memory to store the variance-covariance matrix and much computation time to compute the matrix and its inverse. It is not scalable to large amounts of computer audit data produced by an information system in real time.

Hence, we develop the chi-square multivariate test with less computational overhead. The test statistic for the chi-square multivariate test is:

$$X^2 = \sum_{i=1}^{p} \frac{\left(X_i - \overline{X}_i\right)^2}{\overline{X}_i}. \tag{2-4}$$

In contrast to the T^2 test statistic, the X^2 test statistic does not account for the correlated structure of the p variables. With only the mean vector $\overline{\mathbf{X}}$ in formula (2-4), the chi-square multivariate test detects only the mean shift on one or more of the p variables. Details of the chi-square multivariate test and its application to intrusion detection can be found in [2-54].

When we apply the chi-square multivariate test to intrusion detection, we use the same training data and the same testing data as those for Hotelling's T^2 test. Using 1613 data points of (X_1, X_2, ..., X_{284}) for normal audit events in the training date set, we compute $\overline{\mathrm{X}}$ which characterizes the norm profile. For each of those event types that do not appear in the training data set, we let the average of the variable for that event type take a very small value, 10^{-5} in this study, at the end of training such that the denominators in formula (2-4) are not zero. Using $\overline{\mathrm{X}}$, the X^2 value in formula (2-4) is then computed for each data point of (X_1, X_2, ..., X_{284}) in the testing data set. The computed X^2 value is small if the data point conforms to the norm profile. The ROC curve for the testing results of the chi-square multivariate test is plotted using various signal thresholds on the computed X^2 values for the testing data points, as shown in Figure 2-3.

The comparison of the ROC curves for Hotelling's T^2 test and the chi-square multivariate test reveals better intrusion detection performance of the chi-squared multivariate test than that of

Hotelling's T^2 test. While the chi-square multivariate test detects mainly mean shifts, Hotelling's T^2 test detects both mean shifts and counter-relationships. In fact, Hotelling's T^2 test is more sensitive to counter-relationships than mean shifts because the T^2 test statistic is determined largely by the correlated structure of variables (variance-covariance matrix) [2-53]. Hence, the better intrusion detection performance of the chi-square multivariate test than Hotelling's T^2 test indicates that mean shifts may be more important to intrusion detection than counter-relationships.

The ROC curves for Hotelling's T^2 test and the chi-square multivariate test show better intrusion detection performance of these two anomaly detection techniques than performance of the decision tree based on the single-event representation, even though these two anomaly detection techniques use less data (only normal audit events) during training. This confirms the importance of the frequency property of activity data.

D. Markov Chain and Results

We apply a Markov model that takes into account the ordering property of multiple events for intrusion detection. The application of a Markov model helps answer the question about whether the ordering property of activity data provides additional advantage to intrusion detection, or whether we can detect intrusions from only the frequency property of activity data without the ordering property. Since first-order and high-order Markov models produce comparable intrusion detection performance [2-40 – 2-43], we apply Markov chain – a first-order Markov model that considers only one-step event transitions.

Let X_t be the value of a random variable or the state of a system at time *t*. A Markov chain is a stochastic process with the following assumptions [2-55 – 2-56]:

$$P(X_{t+1}=i_{t+1} \mid X_t=i_t, X_{t-1}=i_{t-1},\ldots,X_0=i_0) = P(X_{t+1}=i_{t+1} \mid X_t=i_t)\text{, and} \tag{2-5}$$

$$P(X_{t+1}=i_{t+1} \mid X_t=i_t) = P(X_{t+1}=j \mid X_t=i) = p_{ij}, \tag{2-6}$$

for all *t* and all states, where p_{ij} is the probability that the system is in a state j at time t+1 given the system is in state t at time t. Formula (2-5) states that the probability distribution of the state at time *t+1* depends on the state at time *t*, and does not depend on the previous states leading to the state at time *t*. Formula (2-6) specifies that a state transition from time *t* to time *t+1* is independent of time.

If the system has a finite number of states, 1, 2, ..., s, the Markov chain model can be defined by a transition probability matrix [2-55 – 2-56]:

$$P = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1s} \\ p_{21} & p_{22} & \cdots & p_{2s} \\ \vdots & \vdots & \vdots & \vdots \\ p_{s1} & p_{s2} & \cdots & p_{ss} \end{bmatrix}, \tag{2-7}$$

and an initial probability distribution:

$$Q = \begin{bmatrix} q_1 & q_2 & \cdots & q_s \end{bmatrix}, \tag{2-8}$$

where q_i is the probability that the system is in state *i* at time 0, and

$$\sum_{j=1}^{j=s} p_{ij} = 1. \tag{2-9}$$

The probability that a sequence of states X_{t-k}, ..., X_t at time *t-k*, ..., *t* occurs in the context of the Markov chain model is computed as follows [2-55 – 2-56]:

$$P(X_{t-k},\cdots,X_t) = q_{x_{t-k}} \prod_{i=k}^{1} p_{X_{t-i}X_{t-i+1}} \tag{2-10}$$

In this study, the transition probability matrix and the initial probability distribution of a Markov chain model are learned from the training data that provide observations of the system state X_0, X_1, X_2, …, X_{N-1} at time t = 0, …, N-1. The transition probability matrix and the initial probability distribution are computed from the training data as follows [2-47]:

$$p_{ij} = \frac{N_{ij}}{N_{i.}} \tag{2-11}$$

$$q_i = \frac{N_i}{N} \tag{2-12}$$

where

N_{ij} is the number of observation pairs X_t and X_{t+1} with X_t in state i and X_{t+1} in state j,

$N_{i.}$ is the number of observation pairs X_t and X_{t+1} with X_t in state i and X_{t+1} in any one of the states 1, …, s,

Ni is the number of Xt's in state i, and

N is the total number of observations.

When we apply the Markov chain to intrusion detection, we use the same training data and the same testing data as those for Hotelling's T^2 test and the chi-square multivariate test. We numbers only time points when audit events occur for time *t*. X_t has 284 possible states representing 284 possible event events at time *t*.

Using the stream of 1613 audit events for normal audit events in the training date set, we compute the transition probability matrix *P* and the initial probability distribution *Q* according to formulas (2-7) and (2-8) which characterize the norm profile. Using *P* and *Q*, we compute the probability that a sequence of the past 15 audit events at time *t* in the testing data, X_{t-14}, …, X_t, occurs in the context of the Markov chain model as follows:

$$P(X_{t-14}, \cdots, X_t) = q_{X_{t-14}} \prod_{i=14}^{1} p_{X_{t-i} X_{t-i+1}} \tag{2-13}$$

Recall that Hotelling's T^2 test and the chi-square multivariate test compute the test statistic based on the past 15 audit events when the smoothing factor is set to 0.3.

The higher probability we obtain from formula (2-13) for an event sequence, the more likely the event sequence is normal. An intrusive event sequence is expected to receive a low probability of support from the Markov chain model of the norm profile.

We assign a small probability of 10^{-5} to initial states and state transitions in the testing data if they have a zero probability value in the transition probability matrix P and the initial probability distribution Q, so that the final result from formula (2-13) is not zero. Details of the implementation can be found in [2-57].

The ROC curve for the testing results of the Markov chain is plotted using various signal thresholds on the computed probability values for event sequences in the testing data, as shown in Figure 2-3.

We compare the ROC curve for the Markov chain based on the ordering property of activity data with the ROC curves for Hotelling's T^2 test and the chi-square multivariate test based on the frequency property of activity data. The comparison reveals slightly better performance of the Markov chain. This indicates that the ordering property of activity data provides some additional advantage than the frequency property to intrusion detection, even using a simple first-order Markov model that considers only one-step event transitions.

V. CONCLUSION

From existing work on intrusion detection, we generalize three properties of activity data in an information system: the frequency property, the duration property and the ordering property. Through a series of studies using the same training data and the same testing data, we provide answers to several questions concerning which properties are necessary to intrusion detection. Our studies show that the frequency property of multiple event types for a given sequence of events is necessary for intrusion detection. A single event at a given time is not sufficient for intrusion detection. Second, the ordering property provides additional advantage than the frequency property to intrusion detection.

Note that intrusive audit data in our studies are "pure" data without white noises from normal activities. Intrusions usually occur in an information system while normal activities are also occurring in the information system. Hence, in real time intrusive audit data are mixed with white noises of normal audit data. For such noisy data, the first-order Markov model of one-step event transitions may not produce good intrusion detection performance. For noisy data, high-order Markov model or event more complex data models may be warranted, which challenges us with the scalability problem of these complex data models. Since the two anomaly detection techniques (Hotelling's T^2 test and the chi-square multivariate test) based on the frequency property provide rather good intrusion detection performance, the frequency property provides a viable tradeoff between computational complexity and intrusion detection performance. When using the frequency property for intrusion detection, a complete data model as in Hotelling's T^2 test detecting both mean shifts and counter-relationships may not be necessary. A simplified data model as in the chi-square multivariate test detecting only multivariate mean shifts may be sufficient. Further studies of these properties of activity data using large amounts of noisy computer audit data are currently ongoing in our laboratory, and will be presented in future reports.

REFERENCES

[2-1] W. Stallings. *Network and Inter-network Security Principles and Practice.* Englewood Cliffs, NJ: Prentice Hall, 1995.

[2-2] C. Kaufman, R. Perlman, and M. Speciner. *Network Security: Private Communication in a Public World.* Englewood Cliffs, New Jersey: Prentice Hall, 1995.

[2-3] T. Escamilla. *Intrusion Detection: Network Security beyond the Firewall.* New York: John Wiley & Sons, 1998.

[2-4] B. Simons. "Building big brother," Communications of the ACM, 43(1), pp. 31-32. January 2000.

[2-5] P. G. Neumann. "Risks of insiders," Communications of the ACM, 42(12), pp. 160, December 1999.

[2-6] M. Godwin. "Net to worry," Communications of the ACM, 42(12), pp. 15-17, December 1999.

[2-7] S. Jajodia, P. Ammann, and C. D. McCollum. "Surviving information warfare attacks," Computer, 32(4), pp. 57-63, April 1999.

[2-8] P. Mann. "Pentagon confronts mounting cyber risks," Aviation Week and Space Technology, 150(12), pp. 82-83, 22 March 1999.

[2-9] B. H. Barnes. "Computer security research: A British perspective," IEEE Software, 15(5), pp. 30-33, September/October 1998.

[2-10] A. Boulanger. "Catapults and grappling hooks: The tools and techniques of information warfare," IBM Systems Journal, 37(1), pp. 106-114, 1998.

[2-11] H. Debar, M. Dacier, and A. Wespi. "Towards a taxonomy of intrusion-detection systems," Computer Networks, 31, pp. 805-822, 1999.

[2-12] R. Lippmann, D. Fried, I. Graf, J. Haines, K., Kendall, D. McClung, D. Weber, S. Webster, D. Wyschogrod, R. Cunningham, and M. Zissman. "Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation." *In Proceedings of the DARPA Information Survivability Conference and Exposition.* Los Alamitos, CA: IEE Computer Society, pp. 12-26, January, 2000.

[2-13] D. Schnackenberg, K. Djahandari. "Infrastructure for intrusion detection and response," *In Proceedings of the DARPA Information Survivability Conference and Exposition.* Los Alamitos, CA: IEE Computer Society, pp. 3-11, January, 2000.

[2-14] T. Bass. "Intrusion detection systems and multi-sensor data fusion," Communications of the ACM, 43(4), pp. 99-105, April 2000.

[2-15] M. Stillerman, C. Marceau, and M. Stillman. "Intrusion detection for distributed applications," Communications of the ACM, 42(7), pp. 62-69, July 1999.

[2-16] U. Lindqvist, and P. A. Porras. "Detecting computer and network misuse through the production-based expert system toolset (P-BEST)," In Proceedings of the 1999 IEEE Symposium on Security and Privacy, IEEE, Oakland, CA, May 1999..

[2-17] P. A. Porras, and P. G. Neumann. "EMERALD: Event monitoring enabling responses to anomalous live disturbances," In Proceedings of NISSC, October 1997.

[2-18] P. G. Neumann, and P. A. Porras. "Experience with EMERALD to date," In Proceedings of the 1st USENIX Workshop on Intrusion Detection and Network Monitoring, Santa Clara, California, April 1999, pp. 73-80.

[2-19] W. Lee, and S. J. Stolfo. "Data mining approaches for intrusion detection," In Proceedings of the 7th USENIX Security Symposium, San Antonio, Texas, January 1998.

[2-20] W. Lee, S. J. Stolfo, and K. W. Mok. "A data mining framework for building intrusion detection models," In Proceedings of the 1999 IEEE Symposium on Security & Privacy, May 1999.

[2-21] W. Lee, S. J. Stolfo, and K. W. Mok. "Mining audit data to build intrusion detection models," In Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining, New York, NY, August 1998.

[2-22] W. Lee, S. J. Stolfo, and K. W. Mok. "Mining in a data-flow environment: Experience in network intrusion detection," In Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '99), San Diego, August 1999.

[2-23] G. Vigna, and R. Kemmerer. "NetStat: A network-based intrusion detection appoach." In Proceedings of the 14th Annual Computer Security Applications Conference, Scottsdale, Arizona, December 1998, http://www.cs.ucsb.edu/~kemm/netstat.html/.

[2-24] G. Vigna, S. T. Eckmann, and R. A. Kemmerer. "The STAT tool suit," In *Proceedings of the DARPA Information Survivability Conference and Exposition.* Los Alamitos, CA: IEE Computer Society, pp. 46-55, January, 2000.

[2-25] S. Kumar. Classification and Detection of Computer Intrusions. Ph.D. Dissertation, Department of Computer Science, Purdue University, West Lafayette, Indiana, 1995.

[2-26] C. Ko, G. Fink, and K. Levitt. "Execution monitoring of security-critical programs in distributed systems: A specification-based approach." In *Proceedings of the 1997 IEEE Symposium on Security and Privacy,* pp. 134-144, 1997.

[2-27] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle. "GrIDS – A graph-based intrusion detection system for

large networks," In Proceedings of the 19th National Information Systems Security Conference, October 1996.

[2-28] T. Bowen, D. Chee, M. Segal, R. Sekar, T. Shanbhag, and P. Uppuluri. "Building survivable systems: An integrated approach based on intrusion detection and damage containment," *In Proceedings of the DARPA Information Survivability Conference and Exposition, Volume II.* Los Alamitos, CA: IEE Computer Society, pp. 84-99, January, 2000.

[2-29] D. E. Denning. "An intrusion-detection model," IEEE Transactions on Software Engineering, SE-13(2), pp. 222-232, February 1987.

[2-30] D. Anderson, T. Frivold, and A. Valdes. *Next-generation Intrusion Detection Expert System (NIDES): A Summary.* Technical Report SRI-CSL-97-07. Menlo Park, CA: SRI International, May, 1995.

[2-31] H. S. Javitz, and A. Valdes. "The SRI statistical anomaly detector." In *Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy.* May 1991.

[2-32] H. S. Javitz, and A. Valdes. *The NIDES Statistical Component Description of Justification.* Technical Report A010. Menlo Park, CA: SRI International, March, 1994.

[2-33] Y. Jou, F. Gong, C. Sargor, X. Wu, S. Wu, H. Chang, and F. Wang. "Design and implementation of a scalable intrusion detection system for the protection of network infrastructure." In *Proceedings of the DARPA Information Survivability Conference and Exposition.* Los Alamitos, CA: IEE Computer Society, pp. 69-83, 2000.

[2-34] W. DuMouchel, M. Schonlau. "A comparison of test statistics for computer intrusion detection based on principal components regression of transition probabilities," In Proceedings of the 30th Symposium on the Interface: Computing Science and Statistics.

[2-35] H. Debar, M. Becker, D. Siboni. "A neural network component for an intrusion detection system," In Proceedings of the 1992 IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, CA, May 1992, pp. 240-250.

[2-36] A. K. Ghosh, A. Schwatzbard, and M. Shatz. "Learning program behavior profiles for intrusion detection." In *Proceedings of the 1st USENIX Workshop on Intrusion Detection and Network Monitoring,* Santa Clara, California, April, 1999, http://www.rstcorp.com/~anup/.

[2-37] S. Forrest, S. A. Hofmeyr, and A. Somayaji. "Computer immunology." *Communications of the ACM,* 40(10), pp. 88-96, October, 1997.

[2-38] H. Debar, M. Dacier, M. Nassehi, and A. Wespi. "Fixed vs. variable-length patterns for detecting suspicious process behavior," In Proceedings of the 5th European Symposium on research in Computer Security, Louvain-la-Neuve, Belgium, September 16-18, 1998, pp. 1-15.

[2-39] C. Warrender, S. Forrest, and B. Pearlmutter. "Detecting intrusions using system calls: Alternative data models," In Proceedings of the 1999 IEEE Symposium on Security and Privacy, pp. 133-145.

[2-40] W. DuMouchel. "Computer intrusion detection based on Bayes factors for comparing command transition probabilities," National Institute of Statistical Sciences, Technical Report No. 91, http://www.niss.org/downloadabletechreports.html.

[2-41] W.-H. Ju, and Y. Vardi. "A hybrid high-order Markov chain model for computer intrusion detection," National Institute of Statistical Sciences, Technical Report No. 92, http://www.niss.org/downloadabletechreports.html.

[2-42] M. Schonlau, W. DuMouchel, W.-H. Ju, A. F. Karr, M. Theus, and Y. Vardi. "Computer intrusion: Detecting masquerades," National Institute of Statistical Sciences, Technical Report No. 95, http://www.niss.org/downloadabletechreports.html.

[2-43] S. L. Scott. "Detecting network intrusion using a Markov modulated nonhomogeneous Poisson process," http://www-rcf.usc.edu/~sls/fraud.ps.

[2-44] P. E. Utgoff, N. C. Berkman, and J. A. Clouse. "Decision tree induction based on efficient tree restructuring," *Machine Learning*, 10, pp. 5–44, 1997.

[2-45] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*, Wadsworth Inc., 1984.

[2-46] S. R. Safavian, and D. Landgrebe. "A survey of decision tree classifier methodology," *IEEE Transactions on Systems, Man and Cybernetics*, 21, pp. 660–674, 1991.

[2-47] T. M. Mitchell, *Machine Learning*. Boston, MA: McGraw-Hill, 1997.

[2-48] SPSS Inc. *AnswerTree 2.0: User's Guide*. Chicago, Illinois: SPSS Inc.

[2-49] N. Ye, X. Li, and S. M. Emran. "Decision trees for signature recognition and state classification," In Proceedings of the IEEE SMC Information Assurance and Security Workshop, West Point, New York, June 2000, pp. 189-194.

[2-50] R. A. Johnson, and D. W. Wichern. *Applied Multivariate Statistical Analysis*. Upper Saddle river, New Jersey: Prentice Hall, 1998.

[2-51] N. Ye, and S. Vilbert. "The EWMA-EWMV technique for detecting anomalies in information systems." *IEEE Transactions on Reliability*, in review.

[2-52] T. P. Ryan. *Statistical Methods for Quality Improvement*. New York: John Wiley & Sons, 1989.

[2-53] N. Ye, Q. Chen, and S. M. Emran. "Hotelling's T^2 multivariate profiling for anomaly detection," In Proceedings of the IEEE SMC Information Assurance and Security Workshop, West Point, New York, June 2000, pp. 175-181.

[2-54] N. Ye, Q. Chen, and S. M. Emran. "Chi-squared statistical profiling for anomaly detection," In Proceedings of the IEEE SMC Information Assurance and Security Workshop, West Point, New York, June 2000, pp. 182-188.

[2-55] N. Ye. "A Markov chain model of temporal behavior for anomaly detection," In Proceedings of the IEEE SMC Information Assurance and Security Workshop, West Point, New York, June 2000, pp. 166-169.

[2-56] W. L. Winston. *Operations Research: Applications and Algorithms.* Belmont, CA: Duxbury Press, 1994.

[2-57] P. Buttorp. *Stochastic Modeling of Scientific Data.* London: Chapman & Hall, 1995.

Chapter 3 An Anomaly Detection Technique Based on a Chi-square Distance Metric for Intrusion Detection

In this chapter, we give the details of an anomaly detection technique based on a Chi-square distance metric for intrusion detection. The testing results of this Statistical Process Control (SPC) technique are presented.

I. INTRODUCTION

An intrusion into an information system compromises the security (e.g., availability, integrity, and confidentiality) of the information system through a series of events in the information system [3-1 – 3-4]. For example, a denial-of-service intrusion compromises the availability of an information system by flooding a server in the information system with an overwhelming number of service requests to the server over a short period of time, and thus denies or degrades the service to legitimate users. Another intrusion may compromise the integrity and confidentiality of an information system by gaining root privileges and then modifying and stealing information in the information system.

As we increasingly rely on information systems to support critical operations in defense, banking, telecommunication, transportation, electric power and many other systems, intrusions into information systems have become a significant threat to our society with potentially severe consequences. Layers of defense can be set up against intrusions through prevention, detection, reaction, and so on. Some examples of intrusion prevention techniques are firewalls and guards, authentication, and encryption. Intrusion detection identifies intrusions being leaked through the fence of prevention and acting on an information system.

This chapter presents an intrusion detection technique based on a chi-square test statistic. The rational for developing this technique is first presented after reviewing existing intrusion detection techniques. Then the technique is described, and an intrusion detection application is defined. Finally the results of testing the technique are presented and discussed.

II. REVIEW OF EXISTING INTRUSION DETECTION TECHNIQUES

Existing intrusion detection techniques fall in two major categories: signature recognition and signature recognition [3-5 – 3-12]. Signature recognition techniques [3-4, 3-5, 3-9] store the signatures of known intrusion scenarios, match the observed behavior with these intrusion signatures, and signal an intrusion when there is a match. Signature recognition techniques have a limitation in that they cannot detect novel attacks whose signatures are unknown.

The limitation of signature recognition techniques can be overcome by using anomaly detection techniques as a complement. For a subject (user, file, privileged program, host machine, computer network, etc.) of interest, anomaly detection techniques [3-6 – 3-12] establish a profile of the subject's normal behavior (norm profile), compare the observed behavior of the subject with its norm profile, and signal an intrusion when the subject's observed behavior departs from its norm profile. Hence, anomaly detection techniques can detect both known and novel intrusions if they demonstrate departures from a norm profile.

Intrusive behavior often shows departures (anomalies) from normal behavior in an information system. For example, in a denial-of-service intrusion through flooding a server, the intensity of events to the server is much higher than the event intensity in a normal operation condition. In an intrusion through gaining root privileges, actions that an intruder takes to get into the information system and maneuver inside the information system are often different from actions of legitimate users in a normal operation condition. Hence, anomalies can be used to detect possible intrusions.

In the existing anomaly detection techniques, a norm profile is specified in one of the following forms: strings [3-7], formal logic [3-8], production rules [3-9], and statistical distributions [3-10 – 3-12]. Forrest and her colleagues [3-7] simulate the human immune system and model the norm profile of a subject as a set of binary strings. For a set of normal strings (self strings), a set of detector strings is constructed so that detector strings do not match self strings. If an incoming string matches any of the detector strings for at least the *r* number of contiguous bits, the detection of an anomaly (a foreign object or non-self) is declared. The string-based anomaly detection technique has been applied to detecting anomalous sequences of system calls to the kernel of a UNIX system. However, this technique has several drawbacks. First, there may exist some nonself strings for which it is impossible to generate detector strings. This issue still remains to be resolved [3-7]. Second, this technique is not robust to noises. Noises in a self string could lead to a match between the self string and one of the detector strings, causing a false alarm. Noises in a non-self string could make the string not matched to any of detector strings, causing a miss. The sensitivity to noises depends heavily on the parameter *r*. On one hand, a small *r* increases the rate of false alarms. On the other hand, a large *r* increases the rate of misses. Currently, the parameter *r* is determined empirically.

The logic-based anomaly detection technique [3-8] has been applied to routers, Domain Name System, and some privileged programs. However, formal logic is difficult for most system administrators to understand and use for specifying a norm profile. In contrast, production rules in expert systems [3-9] are more natural and understandable than formal logic for most system administrators to specify and update a norm profile.

Both logic- and rule-based anomaly detection techniques have a limitation. It is difficult to enumerate and specify all possibilities of normal behavior, especially when multiple subjects (e.g., objects and actions in an information system) are involved. Moreover, the behavior of a subject such as a user is generally not fixed but dynamically changing. It is difficult to specify the dynamically changing behavior in advance using formal logic or production rules.

In the IDES/NIDES systems [3-10 – 3-12], a statistical-based anomaly detection technique is used to represent the expected normal behavior of a subject and variance due to noises. The statistical-based anomaly detection technique overcomes the problems with the string-based, logic-based and rule-based anomaly detection technique in handling noises and variances. However, the statistical technique in IDES/NIDES is a univariate technique that is applied to only one behavior measure, whereas many intrusions involve multiple subjects and multiple actions having impact on multiple behavior measures. Hence, a multivariate anomaly detection technique is needed for intrusion detection.

III. A MULTIVARIATE STATISTICAL TECHNIQUE

Multivariate process control techniques [3-13], such as Hotelling's T^2 [3-14], multivariate cumulative sum (MCUSUM) [3-15], and multivariate exponentially weighted moving average (MEWMA) [3-16], are typically used to monitor and detect anomalies of a process in a manufacturing system. Theoretically, these multivariate process control techniques can be applied to intrusion detection for monitoring and detect anomalies of a process in an information system. Practically, the computationally intensive procedure of these multivariate process control techniques cannot meet the demand of intrusion detection for several reasons. First, intrusion detection must deal with large volumes of high-dimensional process data due to a large number (e.g., hundreds or thousands) of behavior measures and a high frequency of event occurrence. Second, intrusion detection requires a minimum delay of processing each event in an information system to ensure an early indication and warning of intrusions.

For example, let us consider the computational procedure of Hotelling's T^2. Let $\mathbf{X_i} = (X_{i1}, X_{i2}, \ldots, X_{ip})'$ denote an observation of p measures from a process at time i. Assume that when the process is operating normally (in control), the population of **X** follows a multivariate normal distribution with the mean vector μ and the covariance matrix Σ. Using a data sample of size *n*, the sample mean vector $\overline{\mathbf{X}}$ and the sample covariance matrix **S** are usually used to estimate μ and Σ [3-13], where

$$\overline{\mathbf{X}} = (\overline{X_1}, \overline{X_2}, \ldots, \overline{X_p}) \qquad (3\text{-}1)$$

$$\mathbf{S} = \frac{1}{n-1}\sum_{i=1}^{n}(\mathbf{Xi} - \overline{\mathbf{X}})(\mathbf{Xi} - \overline{\mathbf{X}})'. \qquad (3\text{-}2)$$

Hotelling's T^2 statistic for an observation $\mathbf{X_i}$ is determined by the following [3-13]:

$$T^2 = (\mathbf{X_i} - \overline{\mathbf{X}})'S^{-1}(\mathbf{Xi} - \overline{\mathbf{X}}). \qquad (3\text{-}3)$$

A large computed value of T^2 indicates a large deviation of the observation $\mathbf{X_i}$ from the in-control population. We can obtain a transformed value of the T^2 statistic, $\frac{n(n-p)}{p(n+1)(n-1)}T^2$ which follows an *F* distribution with *p* and *n-p* degrees of freedom, by multiplying T^2 by the constant $\frac{n(n-p)}{p(n+1)(n-1)}$. If the transformed value of the T^2 statistic is greater than the tabulated *F* value for a given level of significance, α, then we reject the null hypothesis that the process is in control (normal) and thus signal that the process is out of control (anomalous).

The computation in formula (3-3) involves a covariance matrix and its inverse. Even modern computers have difficulty in storing a large covariance matrix for hundreds or thousands of variables in the memory. Moreover, there may be hundreds or thousands of events occurring in an information system within a short period of time. If we compute the Hotelling's T^2 statistic for all events that occur at a high frequency, the computation time becomes unbearable, and the processing delay of computing the covariance matrix and its inverse for each event becomes unacceptable.

Therefore, a multivariate anomaly detection technique with a low computation cost is needed for intrusion detection. Since the Hotelling's T^2 statistic is a measure of the statistical distance from an observation to the mean estimate of the multivariate normal distribution, we develop a distance measure based on a chi-square test statistic as follows [3-17]:

$$X^2 = \sum_{i=1}^{n} \frac{(X_i - E_i)^2}{E_i} \qquad (3\text{-}4)$$

where X_i is the observed value of the *i*th variable, E_i is the expected value of the *i*th variable, and n is the number of variables. X^2 is small if an observation of the variables is close to the expectation. Using $(\overline{X}_1, \overline{X}_2, \ldots, \overline{X}_n)$ as the estimate of the expectation, we obtain:

$$X^2 = \sum_{i=1}^{n} \frac{\left(X_i - \overline{X}_i\right)^2}{\overline{X}_i} \qquad (3\text{-}5)$$

According to the central limit theorem, when the number of variables is large enough (i.e., greater than 30), X^2 as the sum of squared differences between the observed and expected values of those variables has approximately a normal distribution. Hence, the interval of [μ-

$Z_{\alpha/2}\sigma$, $\mu+Z_{\alpha/2}\sigma$] contains (1-α) percent of the possible values of the X^2 population, where μ and σ are the mean and variance of the X^2 population, α is the significance level, and $Z_{\alpha/2}$ is the tabulated value of the standard normal distribution. The mean and standard deviation of the X^2 population can be estimated from the sample data of X^2 by the sample mean $\overline{X^2}$ and the sample standard deviation S_X^2. The in-control limits to detect anomalies can be set to 3 to obtain 3-sigma control limits [3-18 – 3-19], [$\overline{X^2}$-3S_X^2, $\overline{X^2}$+3S_X^2]. Since we are interested in detecting significantly large X^2 values (large differences between observed and expected values) for intrusion detection, we need to set only the upper control limit to $\overline{X^2}$+3S_X^2. That is, if the computed X^2 for an observation is greater than $\overline{X^2}$+3S_X^2, we signal an anomaly.

Unlike the Hotelling's T^2 statistic that is a distance measure taking into account the covariance of multiple variables, the X^2 statistic in formula (3-5) does not consider the relationships of multiple variables in order to simplify the computation. If intrusions into information systems cause only small violations of variable relationships but large departures from the mean in some of multiple variables, the X^2 statistic can be as effective as the Hotelling's T^2 statistic for intrusion detection.

IV. AN APPLCATION

This section describes an intrusion detection application, including the data source, training data, testing data, and representation of the application problem.

A. Data Source

An information system typically consists of host machines (e.g., machines running a UNIX operation system and machines running the Windows NT operating system) and communication links connecting those host machines, forming a network of host machines. Two sources of data have been widely used to capture events in an information system for intrusion detection: network traffic data and audit trail data (audit data). Network traffic data contain data packets traveling over communication links between host machines to capture events over communication links. Audit data capture events occurring on a host machine. In this study, we use audit data from a UNIX-based host machine (specifically a Sun SPARC 10 workstation with the Solaris operating system), and focus on intrusions into a host machine that leave trails in audit data.

The Solaris operating system from the Sun Microsystems Inc. has a security facility, called the Basic Security Module (BSM). BSM supports the monitoring of activities in a host machine by recording security-relevant events. There are over 250 different types of BSM auditable events, depending on the version of the Solaris operating system. Since there are about 284 different types of BSM audit events on our host machine, we consider 284 event types in this study. A BSM audit record for each event contains a variety of information, including the event type, user ID, group ID, process ID, session ID, the system object accessed, and so on. In this study, we extract and use only the event type that was one of the most critical characteristics of an audit event.

B. Training and Testing Data

For intrusion detection, we want to build a long-term profile of normal events, and to compare events in the recent past to the long-term norm profile for detecting a significant departure. Audit data of normal events are required for training the norm profile. In this study, we use a sample of audit data for normal events that is developed by the MIT (Massachusetts Institute of Technology) Lincoln Laboratory. The sample contains a stream of 3019 audit events. We use the first part of the audit data, consisting of 1613 audit events, for training a norm profile. The second part of the audit data, consisting of 1406 audit events, is used for testing.

We also need the audit data of intrusive events for testing. To create the audit data of intrusive events, we use a number of intrusion scenarios that we have collected over years from various information sources. Some examples of the intrusion scenarios are the password guessing, use of symbolic links to gain root privileges, attempts to gain an unauthorized remote access, etc. We run these intrusion scenarios in a random order through separate sessions on our host machine to obtain the audit data of these intrusion sessions. The audit data of those intrusion sessions, containing 1225 audit events, are used for testing. Hence, the testing data contain 1406 audit events for normal events and 1225 audit events for intrusive events.

C. Problem Representation

Activities on a host machine are captured through a continuous stream of audit events, each of which is characterized by the event type. For intrusion detection, we want to build a long-term profile of normal events, and to compare events in the recent past to the long-term norm profile for detecting a significant departure. We define events in the recent past from time *t-k* to time *t* by a vector of (X1, X2, ..., X284) for 284 event types respectively, based on the exponentially weighted moving average technique [3-18 – 3-19]. At time *t*, the audit events in the recent past from time t-k to time t are summarized as follows.

$$X_i(t) = \lambda * 1 + (1-\lambda) * X_i(t-1) \qquad \text{if the audit event at time } t \text{ falls into the } i\text{th event type} \quad (3\text{-}6)$$

$X_i(t) = \lambda * 0 + (1-\lambda) * X_i(t-1)$ if the audit event at time *t* is different from the *i*th event type

where Xi(t) is the observed value of the *i*th variable in the vector of observation at time *t*, λ is a smoothing constant that determines *k* or the decay rate, and *i* = 1, ..., 284. The most recent observation at time t receives a weight of λ, the observation at time *t-1* receives a weight *of* λ(1-λ), and the observation at time t-k receives a weight of $\lambda(1-\lambda)^k$.

In this study, we initialize Xi(0) to 0 for i = 1, ..., 284. We let λ be 0.3 which is a typical value for the smoothing constant. Figure 3-1 shows the decay effect of the smoothing constant 0.3.

Hence, for each audit event in the training and testing data, we obtain a vector of $(X_1, \ldots, X_{284})$. For example, given the following stream of audit events:

t = 0,	1,	2,	3,	......
	EventType3,	EventType8,	EventType1,	......

At t = 0, all variables in the vector of $(X_1, \ldots, X_{284})$ have a value of 0. At time t = 1, X_3 has a value of 0.3 (equal to 0.3*1+0.7*0), and all other variables have a value of 0. At time t = 2, X_3 has a value of 0.21 (equal to 0.3*0+0.7*0.3), X_8 has a value of 0.3 (equal to 0.3*1+0.7*0), and all other variables have a value of 0. At t = 3, X_3 has a value of 0.147 (equal to 0.3*0+0.7*0.21), X_8 has a value of 0.21 (equal to 0.3*0+0.7*0.3), X_1 has a value of 0.3 (equal to 0.3*1+0.7*0), and all other variables have a value of 0.

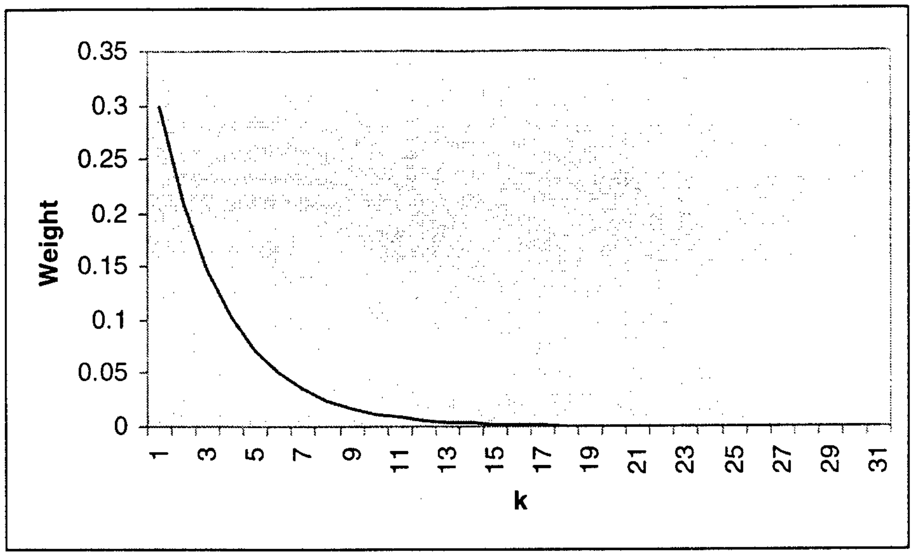


Figure 3-1. The effect of the smoothing constant 0.3.

The expected vector of (X_1, ..., X_{284}) for the long-term profile of normal activities is estimated from the training data by averaging all 1613 vectors that we obtain from 1613 audit events in the training data, to obtain the expected vector, ($\overline{X}_1, \overline{X}_2$, ..., $\overline{X}_{264}$). Considering that events in a host machine actually arrive not all at once but sequentially, we use the following recursive formula to incrementally update $\overline{X}_i$ after each event:

$$\overline{X_{k,i}} = \frac{(k-1)\overline{X_{k-1,i}} + X_{k,i}}{k} \tag{3-7}$$

where k is the index of the current event.

It is possible that some types of audit events do not appear in the training data but occur in the testing data. Hence, the average of the variable for such an event types is zero after the training. To avoid having a zero value for the denominator of formula (5), the average of the variable for such an event type is assigned to a small value after the training. In this study, we used 0.00001.

For each of the audit events (1406 audit events for normal activities and 1223 audit events for attack activities) in the testing data and the corresponding observed vector of (X_1, ..., X_{284}), we compute X^2 as follows:

$$X^2 = \sum_{i=1}^{284} \frac{\left(X_i - \overline{X_i}\right)^2}{\overline{X_i}} \tag{3-8}$$

The computed X^2 is small if the observed vector is close to the expected vector, in other words, the observation is close to the norm profile.

To determine the upper limit of X^2 in terms of $\overline{X^2}+3S_X^2$, we use the computed X^2 values for the first half (first 703 audit events) of 1406 normal events in the testing data to estimate the average ($\overline{X^2}$) and the standard deviation (S_X^2). The upper limit, $\overline{X^2}+3S_X^2$, is then used to determine whether we should signal each of the remaining 703 normal events and 1225 intrusive events in the testing data as an anomaly. If the computed X^2 for an audit event is greater than the upper limit, we signal the audit event as an anomaly.

V. RESULTS AND DISCUSSIONS

Using the computed X^2 values for the first 703 normal events in the testing data, we obtain 1.72E+00, 1.71E+00, and 6.85E+00 for $\overline{X^2}$, S_X^2, and $\overline{X^2}+3S_X^2$ respectively in a scientific expression. That is, the upper limit of the computed X^2 values for normal events is 6.85. If a computed X^2 value for an audit event is greater than 6.85, we signal this audit event as an anomaly.

Figure 3-2 shows the computed X^2 values for the remaining 703 normal events (event no. 1-703) and 1225 intrusive events (event no. 704-1928) in the testing data. Table 3-1 summarizes the statistics (minimum, maximum, average and standard deviation) of the computed X^2 values for the 703 normal events and the 1225 intrusive events. As shown in Figure 3-2 and the statistics in Table 3-1, the computed X^2 values of the normal events are in average smaller than the computed X^2 values of the intrusive events. Note that the smaller the computer X^2 is, the closer the audit event is to the norm profile.

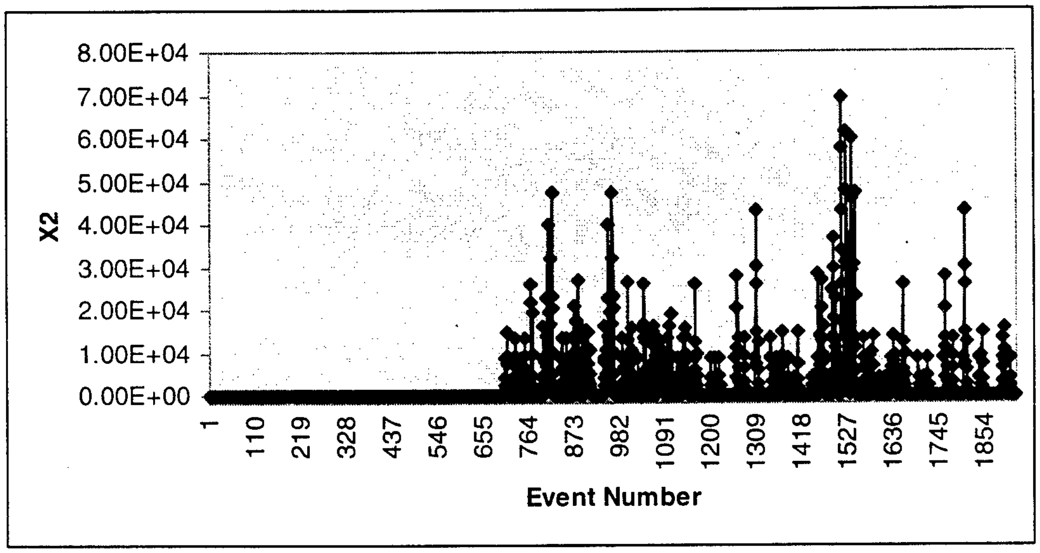


Figure 3-2. The computed X^2 values of audit events in the testing data.

When we use the upper limit of 6.85 to examine the computed X^2 values for the 703 normal events, we obtain no signals. This indicates that there are no false alarms, in other words, we obtain the 0% false alarm rate. A false alarm is a signal for a normal event.

Table 3-1. The statistics of the computed X^2 values for normal and attack data in the testing.

Testing Data	Minimum	Maximum	Average	Standard Deviation
Normal data	5.29E-01	4.06E+00	1.56E+00	9.23E-01
Attack data	6.81E-01	6.92E+04	3.79E+03	7.77E+03

When we use the upper limit of 6.85 to examine the computed X^2 values for the 1225 intrusive events, we obtain 873 signals. The detection rate by audit event is 75% (equal to 873/1225) for the 1225 intrusive events. Since an intrusion session corresponds to one intrusion scenario, we group the 1225 intrusive events into individual intrusion sessions. As a result, there are signals for each individual intrusion session. The minimum detection rate by audit event for all the intrusion sessions is 35%, and the maximum detection rate by audit event for all the intrusion sessions is 86%. Hence, there are signals for each intrusion session. The detection rate by intrusion session is 100%. The 71 percent of the intrusion sessions are detected at the first audit event. That is, the first audit event of these intrusion sessions is signaled. The remaining 29 percent of the intrusion sessions are detected at the second audit event. Hence, every intrusion session is detected at a very early stage.

The 75% detection rate by audit event for intrusions is most likely attributed to the fact that intrusions involve some behavior (e.g., commands) that is also common in normal activities. Hence, we should not expect that every audit event in an intrusion session be detected. However, the overall behavior of intrusions seems different statistically from the overall behavior of normal activities, as we observe in this study from the difference between 0% detection rate by audit event for normal activities and 75% detection rate by audit event for intrusions. It is also possible that some behavior in intrusions may occasionally occur in normal activities, causing false alarms, although such a phenomenon does not show up in the testing data of this study.

Therefore, we expect that more accurate detection results can be achieved at an aggregate behavior level where dominant behavior emerges and persists despite of noises, rather than at the level of individual audit events. An aggregate behavior level can be the level of a session as in this study. For example, in this study we calculate the detection rate by session. Since the maximum detection rate for normal sessions is 0% and the minimum detection rate for intrusion sessions is 35%, we can select any detection rate by session in the range of (0%, 35%) as the decision threshold (e.g., 20%). Using this decision threshold, we are able to clearly distinguish intrusion sessions from normal sessions with the 100% detection rate and the 0% false alarm rate.

On the other hand, it is also desirable to detect an intrusion session early on while it is in progress rather than waiting until the end of the session to get the detection rate by session. A balance between accurate detection and early detection can be achieved by setting three levels of intrusion detection: the normal level (green light), the alert level (yellow light), and the alarm level (red light). When we receive the first signal in a session, we trigger an alert on this session and change the color of the detection light from green to yellow, calling for the attention of the system administrator. We calculate the accumulated detection rate from the first audit event to the current audit event for the session. When the accumulated detection rate up to the current audit event of the session exceeds a decision threshold, we trigger an alarm and change the color of the detection light from yellow to red, claiming the session as an intrusion. Hence, an alert does not become an alarm until the accumulated detection rate up to the current audit event of a session exceeds a decision threshold.

The promising performance of the X^2 statistic as a simplified distance measure for intrusion detection indicates that intrusions may not manifest as violations of variable relationships but large departures from the mean in some variables. A report [3-20] on an application of the Hotelling's T2 statistic to intrusion detection reveals that the performance of the Hotelling's T2 statistic is similar to the performance of the X^2 statistic in this study.

In summary, the results of this study show that the multivariate statistical technique based on the chi-square test statistic achieved the 0% false alarm rate and the 100% detection rate by session. All intrusion sessions are detected at the first or second audit event. Although in this study the multivariate statistical technique is tested using a small set of the testing data, the results of this study demonstrate the very promising potential of this technique for intrusion detection. This technique will be tested in the future with a large set of testing data for further evaluating the performance and scalability.

REFERENCES

3-1. Stallings M. *Network and Inter-network Security Principles and Practice.* Prentice Hall: Englewood Cliffs, NJ, 1995.

3-2. Kaufman C, Perlman R, Speciner M. *Network Security: Private Communication in a Public World.* Prentice Hall: Englewood Cliffs, New Jersey, 1995.

3-3. Ye N, Giordano J, Feldman J. Detecting information warfare attacks: Current state of the art from a process control viewpoint. *Communications of the ACM,* in press.

3-4. Escamilla T. *Intrusion Detection: Network Security beyond the Firewall.* New York: John Wiley & Sons, 1998.

3-5. Lippmann R, Fried D, Graf I, Haines J, Kendall K, McClung D, Weber D, Webster S, Wyschogrod D, Cunningham R, Zissman M. Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation. In *Proceedings of the DARPA Information Survivability Conference and Exposition.* Los Alamitos, CA: IEE Computer Society, pp. 12-26, January, 2000.

3-6. Ghosh AK, Schwatzbard A, Shatz M. Learning program behavior profiles for intrusion detection." In *Proceedings of the 1st USENIX Workshop on Intrusion Detection and Network Monitoring,* Santa Clara, California, April, 1999, http://www.rstcorp.com/~anup/.

3-7. Forrest S, Hofmeyr SA, Somayaji A. Computer immunology. *Communications of the ACM* 1997 **40**(10):88-96.

3-8. Ko C, Fink G, Levitt K. Execution monitoring of security-critical programs in distributed systems: A specification-based approach. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy,* pp. 134-144, 1997.

3-9. Anderson D, Frivold T, Valdes A. *Next-generation Intrusion Detection Expert System (NIDES): A Summary.* Technical Report SRI-CSL-97-07. Menlo Park, CA: SRI International, May, 1995.

3-10. Javitz HS, Valdes A. The SRI statistical anomaly detector. In *Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy.* May 1991.

3-11. Javitz HS, Valdes A. *The NIDES Statistical Component Description of Justification.* Technical Report A010. Menlo Park, CA: SRI International, March, 1994.

3-12. Jou Y, Gong F, Sargor C, Wu X, Wu S, Chang H, Wang F. Design and implementation of a scalable intrusion detection system for the protection of network infrastructure. In *Proceedings of the DARPA Information Survivability Conference and Exposition.* Los Alamitos, CA: IEE Computer Society, pp. 69-83, 2000.

3-13. Ryan TP. *Statistical Methods for Quality Improvement.* John Wiley & Sons: New York, NY, 2000.

3-14. Hotelling H. Multivariate quality control. In Eisenhart C, Hastay MW, and Wallis WA, eds. *Techniques of Statistical Analysis.* McGraw-Hill: New York, NY, 1947.

3-15. Woodall WH, Ncube MM. Multivariate CUSUM quality-control procedures. *Technometrics* 1985 **27**: 285-192.

3-16. Lowry CA, Woodall WH, Champ CW, Rigdon, SE. Multivariate exponentially weighted moving average control chart. *Technometrics* 1992 **34**: 46-53.

3-17. Daniel WW. Biostatistics: A Foundation for Analysis in the Health Sciences. John Wiley & Sons: New York, NY, 1987.

3-18. Montgomery DC. *Introduction to Statistical Quality Control.* John Wiley & Sons: New York, NY, 2000.

3-19. Banks J. *Principles of Quality Control.* John Wiley & Sons: New York, NY, 1989.

3-20. Ye N, Chen Q, Emran SM, Vilbert S. A multivariate quality control technique for cyber intrusion detection. *IEEE Transactions on Computers*, in review.

Chapter 4 Robustness of Chi-square and Canberra Distance Metrics for Intrusion Detection

We develop two multivariate quality control techniques based on chi-square and Canberra distance metrics respectively to detect intrusions by building a long-term profile of normal activities in the information system (norm profile) and using the norm profile to detect anomalies. We investigate the robustness of these two distance metrics by comparing their performance on a number of data sets involving different noise levels in data. This chapter describes our work on comparing different statistical distance metrics in their effectiveness of detecting intrusions when different levels of noises are presented in computer activity data.

I. INTRODUCTION

As we increasingly rely on information infrastructures to support critical operations in defense, banking, telecommunication, transportation, electric power and many other systems, intrusions into information systems have become a significant threat to our society with potentially severe consequences [4-1]. An intrusion compromises the security (e.g., availability, integrity, and confidentiality) of an information system through various means, including denial-of-service, remote-to-local, user-to-root, information probing, and so on [4-2]. Denial-of-service intrusions make a host or network service unavailable by overloading or disrupting the service. Remote-to-local intrusions gain unauthorized access to a host machine without a legitimate user account on the host machine. User-to-root intrusions happen when a regular user on a host machine obtains privileges normally reserved for a root or super user. Information probing intrusions use programs to scan a network of computers for gathering information or searching for known vulnerabilities.

Layers of defense can be set up against intrusions through prevention, detection, and so on. Firewalls, authentication and cryptography are some of on-line intrusion prevention mechanisms to protect information systems from external intrusions [4-3]. Off-line intrusion prevention efforts focus on methodologies of secure software design and engineering. The on-line prevention mechanisms form a fence around information systems to raise the difficulty level of breaking into information systems. However, the fence can only be raised to such a level that services from information systems are not degraded. Although secure software methodologies will continue to improve, bugs and vulnerabilities in information systems are inevitable due to difficulty in managing the complexity of large-scale information systems during their specification, design, implementation, and installation. Intruders explore bugs and vulnerabilities in information systems to attack information systems. Hence, we expect that some intrusions will be leaked through the fence of prevention and results in intrusions into information systems.

Intrusion detection techniques capture intrusions while they are acting on an information system. Existing intrusion detection techniques fall into two major categories: signature recognition and anomaly detection [4-4 – 4-21]. Signature recognition techniques [4-6 – 4-10], also referred to as misuse detection in some literature [4-4, 4-5], match activities in an information system with signatures of known intrusions, and signal intrusions when there is a match. For a subject (user, file, privileged program, host, network, etc.) of interest, anomaly detection techniques establish a profile of the subject's normal behavior (norm profile), compare the observed behavior of the subject with its norm profile, and signal intrusions when the subject's observed behavior deviates significantly from its norm profile [4-11 – 4-21]. Therefore, anomaly detection techniques rely on a norm profile, and consider a deviation of the subject's behavior from its norm profile as a symptom of an intrusion. A justification of using anomaly detection for intrusion detection is provided in [4-15]. A detailed review of the existing intrusion detection techniques can be found in [4-4, 4-5, 4-11]

In our previous work [4-12], we present an anomaly detection technique based on multivariate quality control using a chi-square distance metric to monitor computer audit data of activities in an information system and detect intrusive activities. This technique is developed to overcome: 1) the drawbacks of other statistical-based anomaly detection techniques for intrusion detection, e.g., the sensitivity to the normality assumption, and the limitation to univariate analysis, and 2) the scalability problem of conventional multivariate quality control techniques such as Hotelling's T^2 test when it is applied to large amounts of computer audit data for intrusion detection. Our previous work demonstrates the feasibility and promising performance of our technique based on chi-square distance metric for intrusion detection on one set of computer audit data including both normal and intrusive activities. This paper reports our further work on the robustness of this multivariate quality control technique based on the chi-square distance metric as well as the Canberra distance metric to noises that typically present in computer audit data.

In the following sections, we first define the Chi-square distance metric and the Canberra distance metric. We then describe a number of data sets involving different noise levels to test the robustness of these distance metrics to noises in data. Finally we present and discuss the testing results.

II. DISTANCE METRICS

Hotelling's T^2 test [4-22] is a common multivariate quality control technique to detect anomalies – deviations from a norm profile - especially mean shifts and counter-relationships in a process measured by multiple variables. Hence, Hotelling's T^2 test can be applied as an anomaly detection technique for intrusion detection. Hotelling's T^2 test uses the following statistical distance to measure the deviation of a k-dimensional observation, $\mathbf{x} = [x_1, x_2, \ldots, x_k]'$, from a multivariate normal distribution of the process in control (a norm profile) which is characterized by a sample mean vector, $\bar{\mathbf{x}} = (\overline{x_1}, \overline{x_2}, \ldots, \overline{x_k})$, and a sample variance-covariance matrix, $\mathbf{S}^{-1} = \frac{1}{n-1}\sum_{j=1}^{n}(\mathbf{x}-\bar{\mathbf{x}})(\mathbf{x}-\bar{\mathbf{x}})$:

$$T^2 = (\mathbf{x}-\bar{\mathbf{x}})' \mathrm{S}^{-1} (\mathbf{x}-\bar{\mathbf{x}}) \tag{4-1}$$

where n is the size of the sample.

Computer audit data of activities in an information system often involve hundreds of variables whose values change frequently. To monitor hundreds of variables from computer audit data at a high frequency of sampling, the computation of the variance-covariance matrix and its inverse in formula (4-1) requires a large amount of computer memory and computation time, which becomes unacceptable for real-time monitoring and intrusion detection.

In our previous work [4-12], we introduce the following chi-square distance metric to overcome the scalability problem of the statistical distance metric in Hotelling's T^2 test:

$$\chi^2 = \sum_{i=1}^{k} \frac{(x_i - \overline{x_i})^2}{\overline{x_i}}. \tag{4-2}$$

The chi-square distance measures the deviation from an observation from a norm profile characterized by a sample mean vector only, with the distance scaled simply by the sample mean vector rather than by the complex variance-covariance matrix.

When k is large (e.g., greater than 30), the chi-square distance metric has a normal distribution according to the central limit theorem [4-23]. Hence, the 3-sigma control limits on the chi-square distance metric can be set to $\left[\overline{\chi^2} - 3S_{\chi^2}, \overline{\chi^2} + 3S_{\chi^2}\right]$ for detecting anomalies.

Although the chi-square distance metric can detect only mean shifts, our testing results in [4-12] demonstrate that the multivariate quality control technique based on the chi-square distance metric produces a comparable performance of intrusion detection to Hotelling's T^2 test based on the statistical distance metric [4-14]. It is possible that for intrusion detection mean shifts may be more important than counter-relationships that are not addressed in the chi-square distance metric.

There are also many other distance metrics [4-24]. We present a few representative distance metrics here. The Euclidean distance between an observation and a sample mean vector $\mathbf{x}$ is:

$$d(\mathbf{x},\bar{\mathbf{x}}) = \sqrt{\sum_{i=1}^{k}\left(x_i - \overline{x_i}\right)} = \sqrt{(\mathbf{x}-\bar{\mathbf{x}})'(\mathbf{x}-\bar{\mathbf{x}})} \tag{4-3}$$

Another distance metric is the Minkowski metric

$$d(\mathbf{x},\bar{\mathbf{x}}) = \left[\sum_{i=1}^{k}\left|x_i - \overline{x_i}\right|^m\right]^{1/m} \tag{4-4}$$

For m = 1, $d(\mathbf{x}, \bar{\mathbf{x}})$ measures the "city-block" distance between two points in k-dimensions. For m = 2, $d(\mathbf{x}, \bar{\mathbf{x}})$ becomes the Euclidean distance. In general, varying m changes the weight given to larger and smaller differences.

Two additional distance metrics are the Canberra metric and the Czekanowski coefficient. Both of them are defined for nonnegative variables only. The Canberra metric and the Czekanowski coefficient are given by equations (4-5) and (4-6) respectively.

$$d(\mathbf{x}, \bar{\mathbf{x}}) = \sum_{i=1}^{k} \frac{|x_i - \bar{x}_i|}{(x_i + \bar{x}_i)} \tag{4-5}$$

$$d(\mathbf{x}, \bar{\mathbf{x}}) = 1 - \frac{2\sum_{i=1}^{k} \min(x_i, \bar{x}_i)}{\sum_{i=1}^{k} (x_i + \bar{x}_i)} \tag{4-6}$$

Among these distance metrics, the Euclidean distance metric and the Minkowski distance metric do not scale each dimension when calculating the distance. Our initial testing of the Euclidean distance metric on the same data set as in [4-12] shows the poor performance of the Euclidean distance metric for intrusion detection. The Canberra distance metric is most similar to the chi-square distance metric in that both of them use the scaled distance and the sample mean vector as one scaling factor in the denominator of formula (4-2) and formula (4-5). The Canberra distance adds the observation value as another scaling factor (a part of the denominator). Hence, we carry out this study to compare the performance of the chi-square distance with that of the Canberra distance. We also want to investigate and test the robustness of these distance metrics under different noises levels in data, as noises typically present in computer audit data collected from an information system (discussed in the next section).

III. AN APPLICATION

This section describes the process of applying the Chi-square and Canberra techniques to intrusion detection. It describes the data source, sets of data used, and problem representation.

A. Data Source

A computer and network system within an organization typically includes a number of host machines (e.g., machines running a UNIX operation system and machines running the Windows NT operating system) and communication links connecting those host machines. Currently two sources of data have been widely used to capture activities in a computer and network system for attack detection: network traffic data and host audit trail data (audit data). Network traffic data contain data packets traveling over communication links between host machines to capture activities over communication networks. Audit data capture activities occurring on a host machine.

In this study, we use audit data from a UNIX-based host machine (specifically a Sun SPARC 10 workstation with the Solaris operating system), and focus on intrusions into a host machine that leave trails in audit data. The Solaris operating system from the Sun Microsystems Inc. has an auditing facility, called the Basic Security Module (BSM). BSM monitors the events related to the security of a system and records the crossing of instructions executed by the processor in the user space and instructions executed in the kernel. This is based on the assumption that the actions in the user space cannot harm the security of the system and the security-related actions that can impact the system only take place when users request services from the kernel. BSM records the execution of system calls to the kernel by all processes launched by the users and system programs. A full system call trace gives us an overwhelming amount of information, whereas the audit trail provides a limited abstraction of the same information in which the context switches, memory allocation, internal semaphores and consecutive file reads do not appear. And there is always a straightforward mapping of audit events to system calls.

The BSM audit records contain the detailed information about the audit events in the system. It includes the event type, detailed user and group identification - from the login identity to the one under which the system call is executed, the parameters of the system call execution - file names including full path, command line arguments etc., the return code from the execution, and the error code. However, we use only the event type information contained in it, as many studies [4-11 – 4-18, 4-20] have shown the effectiveness of such information alone for intrusion detection.

There are 284 different types of events in BSM audit data. In UNIX, there are thousands of commands available. Since the audit events are closer to the core of the operating system, the event type is more representative than the actual command sequences used. For example, we can use any text editor, such as *vi* and *ed* to edit a file, but most of time the audit event stream contains the following event types: AUE_EXECVE, AUE_OPEN_R, AUE_ACCESS, AUE_STAT.

In an intrusion session an intruder tries to hide the not-so-frequently-used commands essential for the intrusion by adding large amount of frequently used commands. The effect of this is that intrusion detection techniques recognize these as noises and fails to detect the intruder's attempts. By using the event type information we are able to extract out the redundant information about which particular commands are used, which particular files are accessed etc., and can focus on which particular kernel-level activities (event types) are being used by the intruder. We also avoid the problem of having inconsistent data when we combine normal and intrusion sessions from different sources such as different host machines. As long as they belong to the same Solaris BSM, we get the consistent event type information.

Intrusions usually occur on a host machine while normal activities are also occurring on the host machine. If we consider intrusive activities are signals that we want to detect, normal activities are "white noises" in the background. Hence, computer audit data collected from the host machine contain a mixture of normal and intrusive activities - signals plus noises. The number of normal activities is usually greater than the number of intrusive activities. Intrusion detection is to capture signals from noisy data. The robustness of an intrusion detection technique enables the technique to detect signals even when large amounts of noises are present.

B. Data Sets

In this study we have used four data sets. Among them, the size of the first three data sets is small, containing computer audit data of both normal and intrusive events for a few minutes. The size of the fourth data set is large, containing computer audit data of both normal and intrusive events for four days. We implant different noise levels in the three small data sets to help us in understanding the robustness of the multivariate quality control technique based on two different distance metrics. The large data set resembles a collection of computer audit data for a more realistic mixture of normal and intrusive activities in an information system.

A large portion of data in the four data sets comes from the MIT Lincoln Laboratory (LL) which under the sponsorship of the Defense Advanced Research Projects Agency (DARPA) has created compute audit data of normal and intrusive activities for evaluating intrusion detection systems developed in some DARPA programs [4-2]. A testbed of a computer and network system is set up at LL. Normal activities are simulated on this testbed to resemble activities in a real-world computer and network system at a US Air Force base. Intrusive activities are also simulated while normal activities are simulated. Intrusive activities are simulated based on a number of intrusion scenarios behind intrusion incidents that have happened in the past. The DARPA-LL evaluation data consists of the three data sets created in 1998, 1999 and 2000 respectively. These three data sets are accessible only to DARPA contractors and not to general public. Hence, a small sample of the 1998 evaluation data set is also provided for general public access. All these data sets include computer audit data of both normal and intrusive activities.

We use computer audit data of only normal events in the small sample of the DARPA-LL 1998 evaluation data as the computer audit data of normal events in the three small data sets. There are 3019 normal events. Since the nature of the intrusive events included in the small sample of the DARPA-LL 1998 evaluation data is not clearly specified, we simulate seven intrusion scenarios in seven sessions respectively based on our own collection of intrusions scenarios that have been used to attack computer and network systems in the past. The simulation of these intrusion scenarios produces computer audit data of 1225 intrusive events used in the three small data sets. The seven intrusion sessions produce 215, 225, 54, 36, 413, 247 and 35 intrusive events respectively. The host machine that we use to simulate the intrusion scenarios has the same Solaris BSM. Hence, having normal data and intrusive data from two different host machines is not a concern in this study as discussed in the previous section.

Among the 3019 normal events, the first 2316 normal events are used as the training data for the multivariate quality control technique to build the norm profile that is characterized by the sample mean vector. The remaining 703 normal events are used as the testing data. All the 1225 intrusive events are used as the testing data. Hence, the training data contains the stream of 2316 normal events, and the testing data contains the stream of 703 normal events and the stream of 1225 intrusive events, totally 1928 events.

To create the three small testing data sets with different noise levels for testing the robustness of the two distance metrics, we arrange the testing data in three different ways. The first testing data set is created by putting the stream of the 703 normal events as the first part of the testing data set and then the stream of 1225 intrusive events as the second part of the testing data set. The second testing data set is created by putting the first 400 normal events of the 703 normal events followed by the 530 intrusive events in the first four intrusions sessions, and then the remaining 303 normal events of the 703 normal events followed by the 695 intrusive events in the remaining three intrusion sessions. Hence, in the second testing data set, the normal and intrusive events are inter-mixed, and the lengths of the normal and intrusion periods are nearly same. The third testing data set is created by putting 100 normal events before each intrusion session. Therefore, the noise level increases from the first testing data test (pure normal data and then pure intrusive data), to the second testing data set, and finally to the third testing data set.

We build the large data set from the DARPA-LL 1998 evaluation data that consist of seven weeks (35 days) of training data and two weeks of testing data. We choose four days of data from this training data set as a representative of these 35 days of data to build the training data and the testing data for the large data set. Only four days of data are chosen to reduce the training and testing time. Two weeks of testing data are not considered because the ground truth (normal or intrusive) of events in these weeks of testing data is not provided. Without the ground truth of these events, we cannot evaluate the detection performance of the multivariate quality control technique based on the two distance metrics.

Two days of data are selected where there are not much intrusive activities, and another two days of data are selected where there are lots of intrusive activities. These four days are week-1, Monday data as day-1 data, week-4, Tuesday data as day-2 data, week-4, Friday data as day-3 data and week-6, Thursday data as day-4 data. Table 4-1 summarizes the statistics about these 4 days of data. As we can see from the table, about 3% of day-2 and day-4 events are intrusive events whereas less than 0.80% of events in day-1 and day-3 data are normal events. In terms of sessions, almost one-fourth of the sessions in day-2 and one-twelfth of the sessions in day-3 are intrusive sessions. Day-1 contains mostly normal sessions, and day-4 also does not have many intrusive sessions. There are totally 176 instances of 9 types of intrusions present in these 4 days of data. Day-1 data and day-2 data are used as the training data, and day-3 and day-4 data are used as the testing data.

Table 4-1. Statistics about the large data set.

	Day-1	**Day-2**	**Day-3**	**Day-4**
Event Information				
Number of Events	744085	1320478	2249505	925079
Number of Intrusive Events	3090	36575	16524	31476
Percentage of Intrusive Events	0.42%	2.77%	0.73%	3.40%
Session Information				
Number of Sessions	298	503	339	447
Number of Intrusive Sessions	2	131	29	14
Percentage of Intrusive Sessions	0.67%	26.04%	8.55%	3.13%
Number of Events per Normal Session				
Average	2503	3451	7203	2064
Minimum	1	69	74	96
Maximum	253827	462287	1019443	214705

Number of Events per Intrusion Session				
Average	1545	279	570	2248
Minimum	1101	142	166	1107
Maximum	1989	1737	4986	2841

C. Problem Representation

The Basic Security Module (BSM) of the Solaris operating system monitors 284 different types of audit events. An event can only be one of the 284 event types. We need to transform a stream of audit events into a series of observation vectors in the form of $\mathbf{x} = [x_1, x_2, \ldots, x_k]'$. We have 284 variables for 284 types of audit events respectively, producing $\mathbf{x} = [x_1, x_2, \ldots, x_{284}]'$. Each variable represents the smoothed occurrence frequency of the corresponding event type in the recent past. Therefore, an observation vector is a 284-dimensional vector, $\mathbf{x} = [x_1, x_2, \ldots, x_{284}]'$. The exponentially weighted moving average (EWMA) method [4-25] is used to generate the smoothed occurrence frequency of each event type in the recent past as follows. For the observation value of the nth event in the event stream:

$$\mathbf{x}_n = (x_{1,n}, x_{2,n}, \ldots, x_{284,n}) \tag{4-7}$$

$$x_{i,n} = \lambda \times \theta + (1-\lambda) \times x_{i,n-1};\ x_{i,0} = 0,\ 1 \le i \le 284. \tag{4-8}$$

In formula (4-9), $\mathbf{x}_n$ is the smoothed observation vector for the nth event. The individual dimensional values are computed using formula (8), where $x_{i,n}$ is the smoothed observation value for event type i for the nth event; θ is an indicator function which is 1 if event type i is present in the nth event, and 0 otherwise; $x_{i,n-1}$ is the smoothed observation value at the $(n-1)$th event; and λ is the smoothing constant ($0 < \lambda < 1$). The smoothing constant is usually set to 0.3 [4-26] which is the value used in this study. Formulas (4-7) and (4-8) are used to obtain the observation vector for each audit event in the training data.

The sample mean vector, $\overline{\mathbf{x}} = (\overline{x}_1, \overline{x}_2, ..., \overline{x}_{284})$, is obtained from the series of observation vectors from the training data of normal events to characterize a long-term profile of observation vectors for normal events (norm profile). We use the following incremental updating formula to compute the sample mean vector from the training data:

$$\overline{x}_{i,n} = \frac{(n-1)\overline{x}_{i,n} + x_{i,n}}{n} \text{ where } \overline{x}_{i,0} = 0,\ 1 \le i \le 284 \tag{4-9}$$

For the observation vector from each audit event in the training data, we use the following formula to compute the chi-square distance metric of this observation vector from the sample mean vector:

$$\chi^2 = \sum_{i=1}^{284} \frac{\left(x_i - \overline{x}_i\right)^2}{\overline{x}_i}. \tag{4-10}$$

Using all the chi-square distance values from the training data, we compute the average and the standard deviation of the chi-square distance values, $\overline{\chi^2}$ and S_{χ^2}, and set the 3-sigma control limits on the chi-square distance metric to $\left[\overline{\chi^2} - 3S_{\chi^2}, \overline{\chi^2} + 3S_{\chi^2}\right]$ for detecting intrusions in the testing data.

For each audit event in the testing data, we use formulas (4-7) and (4-8) to obtain the observation vector and use formula (10) to compute the chi-square distance value. From formula (4-10), we can see that the chi-square distance value is a positive value. The larger the chi-square distance value, the larger the deviation of this event from the norm profile, and the more likely the event is a part of an intrusion. Hence, only the upper control limit $\overline{\chi^2} + 3S_{\chi^2}$ is used to determine whether to signal the event as intrusive. If the chi-square distance value is greater than or equal to the upper control limit, a signal is produced on this event to claim this audit event as intrusive; otherwise, no signal is produced.

For the multivariate control technique based on the Canberra distance, formula (4-10) is replaced with the following formula:

$$C = \sum_{i=1}^{284} \frac{\left| x_{i,n} - \bar{x}_i \right|}{\left(x_{i,n} + \bar{x}_i \right)}. \qquad (4\text{-}11)$$

The 3-sigma upper control limit is set to $\bar{C} + 3\, S_C$.

IV. RESULTS AND DISCUSSIONS

This section presents the testing results for the multivariate quality control technique based on the chi-square distance metric and the Canberra distance metric, and discusses the robustness of these distance metrics. We use the Receiver Operating Characteristic (ROC) analysis [4-27 – 4-28] to evaluate the performance of intrusion detection.

For each testing data set, the multivariate quality control technique based on each distance metric produces a set of the distance values (for the chi-square distance metric or the Canberra distance metric) for the audit events in the testing data set. Given a signal threshold, we can compute the false alarm rate and the hit rate. A signal is produced on an event if the distance value for this event is greater than the signal threshold; otherwise, no signal is produced on this event. If a signal is produced on a truly intrusive event, this is a hit. If a signal is produced on a truly normal event, this is a false alarm. Among all the intrusive events in the testing data set, we count how many signals are produced on these events. The hit rate is the ratio of this count to the total number of the intrusive events in the testing data set. Among all the normal events in the testing data set, we count how many signals are produced on these events. The false alarm rate is the ratio of this count to the total number of the normal events in the testing data set. Hence, for a given signal threshold, we obtain a pair of the hit rate and the false alarm rate. By varying the value of the signal threshold, we obtain many pairs of the hit rate and the false alarm rate. These pairs are plotted as a ROC curve. Measuring the hit rate alone indicates only how many intrusive events can be detected. It does not indicate human workload required to analyze false alarms on normal events. A low false alarm rate along with a high hit rate means a desirable performance of intrusion detection in that the detection results can be trusted and human labor required to confirm signals is minimized.

A. Small Data Sets

Figure 4-1 shows the ROC curve for the testing results on the first small testing data set with the lowest noise level (pure normal data followed by pure intrusive data) among the three small testing data sets. Figure 4-2 shows the ROC curve for the testing results on the second small testing data set with the intermediate noise level (including 400 normal events, 530 intrusive events, 303 normal events, and then 695 intrusive events). Figure 4-3 shows the ROC curve for the testing results on the third small testing data set with the highest noise level (about 100 normal events before each intrusion session). The top-left corner of the charts in these figures represents the 100% hit rate and the 0% false alarm rate. Hence, the closer the ROC curve is to the top-left corner, the better the intrusion detection performance.

Figure 4-1 indicates that the Canberra distance metric yields a better performance than the chi-square distance metric in the condition of the lowest noise level. Figure 2 and Figure 3 indicate that the chi-square distance metric yields a better performance than the Canberra distance metric in the conditions of the intermediate noise level and the highest noise level respectively. By comparing the performance of the Canberra distance metric among the three conditions of different noise levels, we find that the performance of the Canberra distance metric drops as the noise level increases. The performance of the Canberra distance metric in the condition of the highest noise level drops almost to the diagonal line in the chart that corresponds to the expected performance of a random detector. Hence, the Canberra distance metric is sensitive to noises. On the other hand, the performance of the chi-square distance metric is consistent among different noise levels. Hence, the chi-square distance metric is robust to noises.

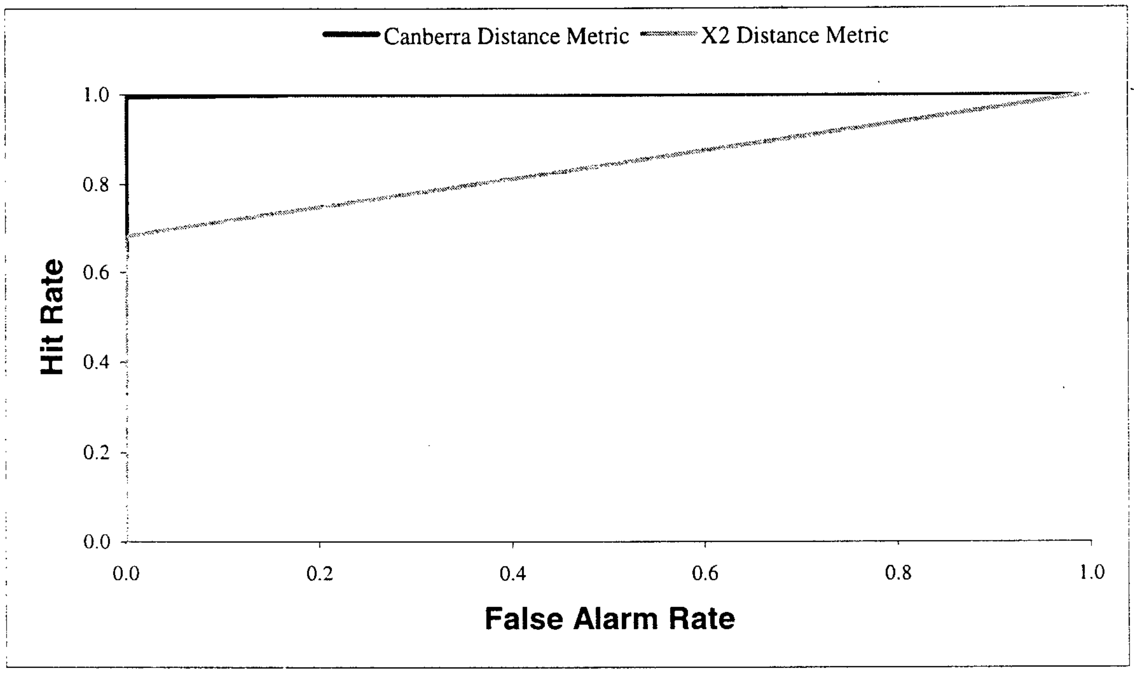


Figure 4-1. ROC curve for the first small data set at the lowest noise level.

At all the three noise levels, the chi-square distance metric produces about the 70% hit rate at the 0% false alarm rate. Since common commands such as *vi* and *ls* in a UNIX operating system are frequently used by both normal users and intruders, we should not expect that we can signal every audit event in an intrusion session, thus having the 100% detection rate at the 0% false alarm rate.

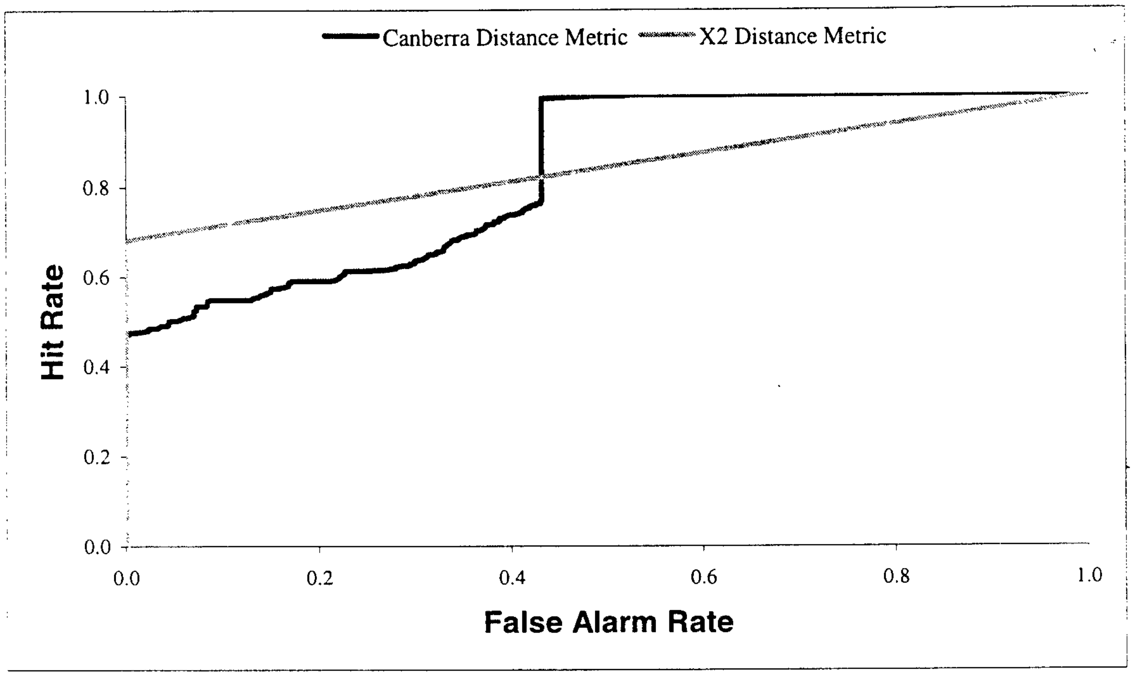


Figure 4-2. ROC curve for the second small data set at the intermediate noise level.

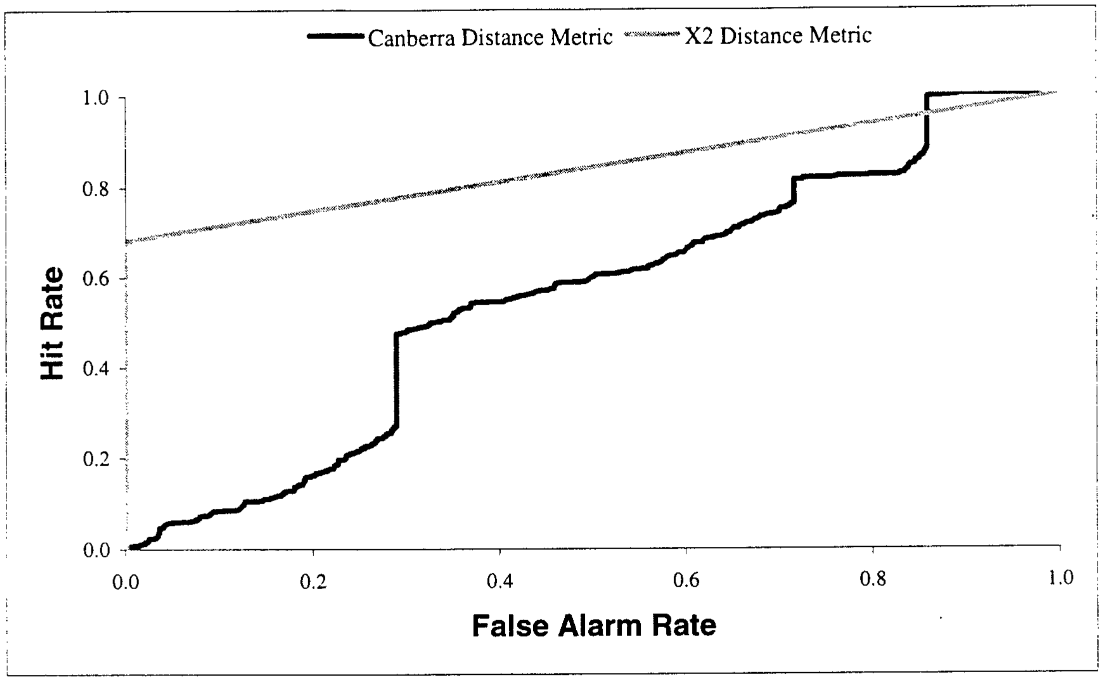


Figure 4-3. ROC curve for the third data set at the highest noise level.

B. Large Data Set

Since the large data set contains a large number of events, we perform the ROC analysis not based on events but based on sessions. We group the events in the large testing data set by session, count how many of the events in each session are signaled, and then compute the ratio of the number of signals to the number of events in that session as the session signal ratio. If it is an intrusion session, we expect a high session signal ratio. If it is a normal session, we expect a low session signal ratio. Figure 4-4 shows the ROC curve of the testing results on the large data set that resembles computer audit data collected from real-work information systems. A ROC curve based on session signal ratio tells us how well we distinguish the intrusion sessions from the normal sessions. The ROC curve in Figure 4-5 indicates that the chi-square distance metric performs better than the Canberra distance metric in distinguishing the intrusion sessions from the normal sessions based on session signal ratio.

V. CONCLUSION

The multivariate quality control technique based on the chi-square distance metric demonstrates consistently better performance under the conditions of different noise levels than the multivariate quality control technique based on the Canberra distance metric. The chi-square distance metric is much more robust to noises in data than the Canberra distance metric. The computation involved in the multivariate quality control technique based on the chi-square distance metric is also simple, which makes it scalable to large amounts of computer audit data for real-time intrusion detection.

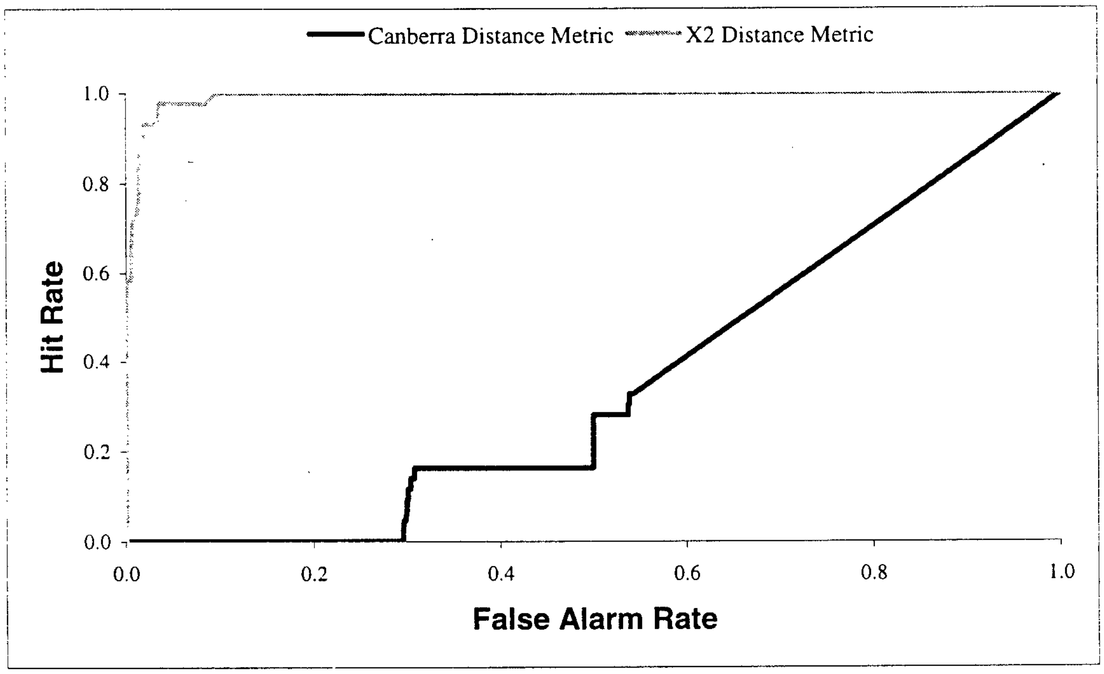


Figure 4-4. ROC curve for the large data set based on session signal ratio.

Intrusions occur while normal activities are also taking place on a host machine. Even if no normal users are using the host machine at the time when an intrusion occurs, there are system programs (e.g., ftp daemon and the program monitoring the inputs from the keyboard) in the information system running all the time. Hence, computer audit data from a host machine always contain noises of normal activities. Because of the scalability and robustness of the multivariate quality control technique based on the chi-square distance metric to noises in large amounts of computer audit data, this technique is highly recommended to be included in commercial intrusion detection systems for real-time intrusion detection. Most of existing commercial intrusion detection systems use a signature recognition technique for intrusion detection. Since signature recognition techniques can detect only known intrusions, an anomaly detection technique such as the multivariate quality control technique based on the chi-square distance metric is highly desirable to complement the signature recognition technique in commercial intrusion detection systems for real-time or off-line intrusion detection.

REFERENCES

4-1. Stallings W. *Network and Inter-network Security Principles and Practice.* Prentice Hall: Englewood Cliffs, NJ, 1995.

4-2. Lippmann R, Fried D, Graf I, Haines J, Kendall K, McClung D, Weber D, Webster S, Wyschogrod D, Cunningham R, Zissman M. Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation. In *Proceedings of the DARPA Information Survivability Conference and Exposition.* IEEE Computer Society: Los Alamitos, CA: IEEE Computer Society, pp. 12-26, January 2000.

4-3. Kaufman C, Perlman R, Speciner M. *Network Security: Private Communication in a Public World.* Prentice Hall: Englewood Cliffs, New Jersey, 1995.

4-4. Debar H, Dacier M, Wespi A. Towards a taxonomy of intrusion-detection systems. *Computer Networks* 1999 **31**: 805-822.

4-5. Escamilla T. *Intrusion Detection: Network Security beyond the Firewall.* John Wiley & Sons: New York, 1998.

4-6. Vigna G, Eckmann S, Kemmerer R. The STAT Tool Suite. In *Proceedings of the DARPA Information Survivability Conference and Exposition.* IEEE Computer Society: Los Alamitos, CA, pp. 46-55, January 2000.

4-7. Kumar S. *Classification and Detection of Computer Intrusions.* Ph.D. Dissertation, Department of Computer Science, Purdue University, West lafayette, Indiana, 1995.

4-8. Lee W, Stolfo SJ, Mok K. Mining in a data-flow environment: Experience in network intrusion detection. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '99),* San Diego, CA, August, 1999, http://www.cs.columbia.edu/~sal/JAM/PROJECT/.

4-9. Anderson D, Frivold T, Valdes A. *Next-generation Intrusion Detection Expert System (NIDES): A Summary.* Technical Report SRI-CSL-97-07. Menlo Park, CA: SRI International, May 1995.

4-10. Neumann P, Porras P. Experience with EMERALD to date. In *Proceedings of the 1st USENIX Workshop on Intrusion Detection and Network Monitoring,* Santa Clara, California, April 1999, pp. 73-80, http://www.csl.sri.com/neumann/det99.html/.

4-11. Ye N, Li X, Chen Q, Emran SM, Xu M. Probabilistic techniques for intrusion detection based on computer audit data. *IEEE Transactions on Systems, Man, and Cybernetics* 2001 **31**(4).

4-12. Ye N, Chen Q. An anomaly detection technique based on a chi-square statistic for detecting intrusions into information systems. *Quality and Reliability Engineering International* 2001 **17**(2): 105-112.

4-13. Ye N, Chen Q, Emran SM, Noh K. Chi-square statistical profiling for anomaly detection. In *Proceedings of IEEE Systems, Man and Cybernetics Information Assurance & Security Workshop*. West Point: United States Military Academy, 2000.

4-14. Ye N, Chen Q, Emran SM, Vilbert S. Hotelling's T^2 multivariate profiling for anomaly detection. In *Proceedings of IEEE Systems, Man and Cybernetics Information Assurance & Security Workshop*. West Point: United States Military Academy, 2000.

4-15. Denning DE. An intrusion-detection model. *IEEE Transactions on Software Engineering* 1987 **SE-13**(2): 222-232.

4-16. Ghosh AK, Schwatzbard A, Shatz M. Learning program behavior profiles for intrusion detection. In *Proceedings of the 1st USENIX Workshop on Intrusion Detection and Network Monitoring,* Santa Clara, California, April 1999, http://www.rstcorp.com/~anup/.

4-17. Javitz HS, Valdes A. The SRI statistical anomaly detector. In *Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy.* May 1991.

4-18. Javitz HS, Valdes A. *The NIDES Statistical Component Description of Justification.* Technical Report A010. Menlo Park, CA: SRI International, March 1994.

4-19. Jou Y, Gong F, Sargor C, Wu X, Wu S, Chang H, Wang F. Design and implementation of a scalable intrusion detection system for the protection of network infrastructure. In *Proceedings of the DARPA Information Survivability Conference and Exposition.* Los Alamitos, CA: IEEE Computer Society, pp. 69-83, January 2000.

4-20. Forrest S, Hofmeyr SA, Somayaji A. Computer immunology. *Communications of the ACM* 1997 **40**(10): 88-96.

4-21. Ko C, Fink G, Levitt K. Execution monitoring of security-critical programs in distributed systems: A specification-based approach. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy,* pp. 134-144, 1997.

4-22. Hotelling H. Multivariate quality control. In Eisenhart C, Hastay MW, and Wallis WA, eds. *Techniques of Statistical Analysis.* McGraw-Hill: New York, NY, 1947.

4-23. Daniel WW. Biostatistics: A Foundation for Analysis in the Health Sciences. John Wiley & Sons: New York, NY, 1987.

4-24. Johnson RA, Wichern DW. *Applied multivariate Statistical Analysis*. Prentice Hall: New Jersey, 1998.

4-25. Montgomery DC. *Introduction to Statistical Quality Control.* John Wiley & Sons: New York, NY, 2000.

4-26. Ryan TP. *Statistical Methods for Quality Improvement.* John Wiley & Sons: New York, 1989.

4-27. Swets JA. The Relative Operating Characteristic in Psychology. *Science* 1973 **182**: 990–1000.

4-28. Egan JP. *Signal detection theory and ROC-analysis.* Academic Press: New York, 1975.

Chapter 5 Robustness of the Markov Chain Model for Intrusion Detection

This chapter describes our work on the robustness of the Markov chain model for intrusion detection when different levels of noises are present in computer activity data.

I. INTRODUCTION

A cyber attack is launched through a series of computer actions to compromise the security (e.g., availability, integrity, and confidentiality) of a computer and network system. As we increasingly rely on information infrastructures to support critical operations in defense, banking, telecommunication, transportation, electric power and many other systems, cyber attacks have become a significant threat to our society with potentially severe consequences [5-1 – 5-10]. Cyber attack detection aims at identifying cyber attacks while they are acting on a computer and network system.

There exist two general approaches to detecting cyber attacks: pattern recognition and anomaly detection, which complement each other in cyber attack detection [5-11 – 5-46].

Pattern recognition techniques identify and store signature patterns of known attacks. They then match the subject's observed behavior with those known patterns of attack signatures and signal attacks when there is a match. Pattern recognition techniques have been used in many commercial software and research prototypes [5-11, 5-16 – 5-26]. Attack signatures have been characterized as strings, event sequences, activity graphs, and attack scenarios (consisting of event sequences, their preconditions and target compromised states). Rules [5-16 – 5-22], state transition diagrams [5-23 – 5-24], colored Petri net models [5-25] and decision trees [5-26] have been used to represent and match patterns of attack signatures. Although pattern recognition techniques are efficient in detecting known attacks, they cannot detect novel or unusual attacks whose signature patterns are unknown. Moreover, the repository of attack signature patterns must be constantly updated to remain useful in a dynamically changing system environment (e.g., configurations, protocols, architectures, and attack scenarios).

For a subject (e.g., a user, file, privileged program, host machine, or network) of interest, anomaly detection techniques, consisting of establishing a norm profile (a profile of the subjects normal activities), observe the activities of the subject and signal attacks when the subject's observed activities differ significantly from its norm profile [5-27 – 5-46]. Anomaly detection techniques consider the deviations of a subject's observed activities from its norm profile, that is, anomalies as symptoms of attacks. Denning [5-30] provides a justification of the anomaly detection approach to cyber attack detection. Anomaly detection techniques can detect both novel and known attacks if they demonstrate significant differences from the norm profile. Since anomaly detection techniques treat all anomalies as attacks, false alarms are expected when anomalies result from irregular behavior instead of cyber attacks. Pattern recognition techniques and anomaly detection techniques can improve detection capabilities when used together.

This chapter presents some work on the anomaly detection approach to cyber attack detection and concentrates on the robustness of a Markov chain technique. Several anomaly detection techniques exist, and differ in the representation of a norm profile and the inference of a deviation from a norm profile.

Specification-based anomaly detection techniques [5-27 – 5-29] describe security policies and authorized activities of a well-defined subject such as a privileged program or a network server in terms of formal logic, rules, and/or graphs. However, it is often difficult to enumerate and specify all possible normal activities or security policies of a subject, especially a large-scale subject such as a host machine or a network where interactions of multiple objects and actions are inevitable. Moreover, activities of many subjects in a computer and network system (e.g., users and files) involve random noises and variations that cannot be well addressed by formal logic, rules, and/or graphs.

Statistical-based anomaly detection techniques [5-30 – 5-36] construct a statistical profile (i.e., a univariate or multivariate statistical distribution) of a subject's normal activities from historic data. These statistical profiles do not consider the order in which the events occur to the subject. Although these statistical-based anomaly detection techniques have demonstrated promising performance for cyber attack detection with respect to detection rate and false alarm rate, it is not clear whether further improvement in attack detection performance can be obtained by taking into account the ordering of events (event transitions). Ye, Li, and Emran [5-26] show that attack detection using a number of consecutive events during a given period produces better performance than using a single event at a given time. Since many cyber attacks require a series of related events to accomplish, it is possible to improve attack detection performance by incorporating the ordering of events.

There exist several anomaly detection techniques that investigate the ordering of events for cyber attack detection [5-38 – 5-47]. Both recurrent and perceptron neural networks have been used to predict the next event (e.g., command or system call) from a series of the past events [5-37 – 5-38]. Immunology-based anomaly detection techniques [5-38 – 5-41] capture a large set of event sequences as the norm profile from historic data of a subject's normal behavior, and use either negative selection or positive selection algorithms to detect incoming event sequences that differ from event sequences in the norm profile. Anomaly detection techniques based on Bayes' parameter estimation for building a Markov model of a norm profile and likelihood ratio tests for detecting anomalies [5-42 – 5-45], Bayesian networks [46], hidden Markov models [5-41], or principal components analysis [5-47], are also investigated. However, the training of all the above techniques taking into account the ordering of events is computationally intensive [5-41, 5-44]. The inference of anomalies in many of the above techniques [5-41, 5-38 – 5-41, 5-42 – 5-45, 5-46] is also computationally intensive. Immunology-based anomaly detection techniques require large amounts of memory to store a large set of event sequences.

A comparative study on a number of anomaly detection techniques indicates that a technique based on a first-order Markov chain model of event transitions as a norm profile produces comparable performance with a high-order Markov model and the other techniques [5-42 – 5-45]. However, this technique based on the first-order Markov chain model of a norm profile is still computationally intensive due to its use of the Bayes' parameter estimation for learning the norm profile and a likelihood ratio test for inferring anomalies. Considering large amounts and high frequency of events in a computer and network system, this technique is not applicable to cyber attack detection in real time.

This chapter examines the robustness of a Markov chain technique to noise as it is introduced into the system. Section II describes the Markov chain model. Section III discusses the representation of the attack detection problem using the Markov chain model and two applications of the technique. Section IV defines the training and testing data and how they are obtained. The results of the applications illustrated in Section III are provided in Section V along with a discussion of these results. Conclusions are presented in Section VI.

II. MARKOV CHAIN MODEL

A discrete-time stochastic process can be described as an arbitrary family of random variables $X(t,\xi,\theta)$ where t is a series of discrete time points, $t_0 < t_1 < \cdots < t_i < \cdots < t_n \in T$ and T is a parametric space. Here ξ represents all the random factors of the system, design parameter θ belongs to an infinite set of all possible design parameters, and the set of all possible values assumed by the process, S, is called the state space. A realization of the discrete-time stochastic process may be described by plotting $X(t)$ against discrete time $t = t_0, t_1 \cdots t_i \cdots, t_n \in T$ when we assign $\theta \in \Theta$ a value.

A Markov chain is a special type of discrete-time stochastic process with the assumption [5-48 – 5-51]

$$P(X_{t+1} = i_{t+1} \mid X_t = i_t, X_{t-1} = i_{t-1}, \ldots, X_0 = i_0) = P(X_{t+1} = i_{t+1} \mid X_t = i_t), \qquad (5\text{-}1)$$

for any $t \in T$ and $i_t \in S$. That is, the probability distribution of the state at time $t+1$ depends on the state at time t, and does not depend on the previous states leading to the state at time t. Therefore, a Markov chain is a first-order Markov model which considers only one-step transition probabilities. A high-order Markov model considers state transitions over more than one time step. If we assume that a state transition from time t to time t+1 is independent of time, the Markov chain becomes a stationary Markov chain [5-24] and is denoted

$$P(X_{t+1} = i_{t+1} \mid X_t = i_t) = P(X_{t+1} = j \mid X_t = i) = p_{ij}, \text{ for all } t \text{ and all states}, \qquad (5\text{-}2)$$

where p_{ij} is the probability that the system is in a state j at time t+1 given the system is in state i at time t. If the system has a finite number of states, 1, 2, ..., s, the Markov chain model can be defined by a transition probability matrix [5-23]

$$P = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1s} \\ p_{21} & p_{22} & \cdots & p_{2s} \\ \vdots & \vdots & \vdots & \vdots \\ p_{s1} & p_{s2} & \cdots & p_{ss} \end{bmatrix}, \tag{5-3}$$

where

$$\sum_{j=1}^{j=s} p_{ij} = 1 . \tag{5-4}$$

The initial probability distribution [5-23] is represented by

$$Q = [q_1 \quad q_2 \quad \cdots \quad q_s] , \tag{5-5}$$

where q_i is the probability that the system is in state i at time 0. The probability that a sequence of states X_1, ..., X_T occurs in the context of the Markov chain model is computed using

$$P(X_1, \cdots, X_T) = q_{x1} \prod_{t=2}^{T} P_{X_{t-1} X_t} \tag{5-6}$$

A logarithmic transformation of Equation (5-6) can be expressed as

$$\log(P(X_1, \cdots, X_T)) = \log(q_{x_1}) \sum_{t=2}^{T} \log(P_{X_{t-1} X_t}) . \tag{5-7}$$

Since we will be working with a very large sequence of events and we are taking the logarithm of the probabilities the central limit theorem can be applied. According to the Central Limit Theorem, we can conclude that the sequence of probabilities we obtained from the Markov chain model conform to the normal distribution with which we are familiar and is easy for us to handle.

The transition probability matrix and the initial probability distribution of a Markov chain model can be obtained from the observations of the system state in the past. Given the observations of the system state $X_0, X_1, X_2, \ldots, X_{N-1}$, we estimate the transition probabilities and the initial probabilities using [5-52, 5-53]

$$p_{ij} = \frac{N_{ij}}{N_{i.}} \tag{5-8}$$

and

$$q_i = \frac{N_i}{N} \tag{5-9}$$

where

- N_{ij} is the number of observation pairs, X_t and X_{t+1}, with X_t in state i and X_{t+1} in state j,
- $N_{i.}$ is the number of observation pairs, X_t and X_{t+1}, with X_t in state i and X_{t+1} in any one of the states $1, \ldots, s$,
- N_i is the number of X_t's in state i, and
- N is the total number of observations.

Bayes parameter estimation can also be used to estimate the transition probability matrix and the initial probability distribution from historic data. However, Bayes' parameter estimation is not adopted in this study due to its high computational cost. Provided with sufficient historic data, the estimation of the transition probability matrix and the initial probability distribution via formulas (5-8) and (5-9) can be quite stable.

III. APPLICATIONS

In this section, we define the attack detection problem, including the data source and problem representation, by way of a particular example. We will further illustrate the robustness properties of the Markov chain technique through a second application.

A. Application 1

A computer and network system within an organization typically includes a number of host machines (e.g., machines running a UNIX operation system and machines running the Windows NT operating system) and communication links connecting those host machines. Currently two sources of data have been widely used to capture activities in a computer and network system for attack detection: network traffic data and audit trail data (audit data). Network traffic data contain data packets traveling over communication links between host machines, and thus capture activities over communication networks. Audit trail data capture activities occurring on a host machine. In this study, we used audit data from a UNIX-based host machine (specifically a Sun SPARC 10 workstation with the Solaris operating system), and focused on attacks on a host machine that left trails in the audit data.

The Solaris operating system from Sun Microsystems Inc. has a security extension called the Basic Security Module (BSM). The BSM extension supports the monitoring of activities on a host by recording security-relevant events. A BSM auditable event falls into one of two categories, kernel events or user-level events. Kernel events are generated by system calls to the kernel of the Solaris operation system while user-level events are generated by application software.

There are more than 250 different types of BSM auditable events, depending on the version of the Solaris operating system. There are also about 284 different types of BSM audit events on the host machine used in this study. A BSM audit record for each event contains a variety of information, including the event type, user ID, group ID, process ID, etc. In this study, we extract and use only the *event type*, one of the most critical characteristics of an audit event. Therefore, activities on a host machine are captured through a continuous stream of audit events, each characterized by the event type.

By considering computer audit data with only event type, we detect attacks on a host machine. When an attack is detected from a series of audit events, we can trace the original audit records for the series of audit events and obtain more information such as the user(s) and process(es) from the original audit records to assist responses to an attack.

Both normal and attack activities on a host machine contain sequences of computer actions with random behavior. Sequences of computer actions induce sequences of audit events. Random behaviors create the uncertainty in predicting the next audit event. Considering the 284 types of audit events as the 284 possible states of a host machine, event sequences on the host machine can be represented as a discrete-time stochastic process. Discrete points in time for the discrete-time stochastic process are defined not by a fixed time interval but by times when audit events take place, because we are mainly interested in event transitions in this study. In another study, we develop a technique that examines the time interval or the event intensity for a given period of time from computer audit data.

For attack detection, we want to build a long-term norm profile of event sequence, and to compare the event sequence in the recent past to the long-term norm profile for detecting a large difference. Using formulas (5-7) and (5-8), we train and build a Markov chain model of event sequence as the long-term norm profile by learning the transition probability matrix and the initial probability distribution from a stream of audit events that is observed during the normal usage of the host machine.

We define the event sequence in the recent past by opening up an observation window of size N on the continuous stream of audit events to view the last N audit events from the current time t, $E_{t-(N-1)}, \ldots, E_t$, where E represents an event. At the next time $t+1$, the observation window contains $E_{t-N+2}, \ldots, E_{t+1}$. We investigated $N = 100$ and $N = 10$ to determine how the window size affects the robustness of our technique and found that $N = 10$ is superior to a window size of $N = 100$. This phenomenon is due in part by the fact that as the window size increases, the opportunity for more false alarms would also increase; thus, diminishing the performance of the Markov chain technique.

For the audit events $E_{t-99}, \ldots, E_t$ in the window at time t, we examine the type of each audit event and obtain the sequence of states $X_{t-99}, \ldots, X_t$ appearing in the window, where X_i is the state (the type of audit event) that the audit event E_i takes. Using formula (5-6), we compute the probability that the sequence of states $X_{t-99}, \ldots, X_t$ occurs in the context of the normal usage; that is, the probability that the Markov chain model of the norm profile supports the sequence of states $X_{t-99}, \ldots, X_t$,

$$P(X_{t-99}, \cdots, X_t) = q_{X_{t-99}} \prod_{i=t-98}^{t} p_{X_{i-1}X_i}$$

The larger the probability obtained, the more likely the sequence of states results from normal activities. A sequence of states from attack activities is expected to receive a low probability of support from the Markov chain model of the norm profile.

It is possible that a sequence of states from a window of the testing data presents an initial state and/or some state transitions which are not encountered in the training and thus have the probabilities of zero in the initial probability distribution or the transition probability matrix of the Markov chain model. While using formula (5-6) to infer the probability of support to the sequence of states, the probabilities of zero would dominate the final probability result from formula (5-6) and make it zero. This would make it impossible to distinguish attack activities from normal activities with noises, since noises in normal activities may introduce a new initial state and/or new state transitions. The amount of the new initial state and/or new state transitions from noises in normal activities is expected to be much less than the amount of the new initial state and/or new state transitions from attack activities.

If we observe the host machine for a very long period of time, we have the opportunity to observe every possible initial state and state transition due to random noises in behavior. However, not every possibility appeared in our training data due to the limited sample size of the data. For example, if the true probability of an initial state is 0.00001 and the sample size of the training data is 1613, the expected number of times that this initial state appears in the training data [5-24] becomes 0.01613 (equal to 0.00001*1613). That is, we should not expect to observe this initial state in the training data, although the true probability of this initial state is not zero.

Therefore, while using formula (5-6) to infer the probability of support to a sequence of states, we replace the probability of zero for a new initial state or a new state transition with a small probability value. By assigning a small probability rather than zero to the new initial state or the new state transition, the small noise effects in normal activities on the final probability result of formula (5-6) becomes distinguishable from the larger effect of attack activities on the final probability result of formula (5-6).

In this study, the sample size of the training data (to be presented in the next section) is 1613. Since 1/1613 yields 0.0006, any initial state or state transition with a true probability less than 0.0006 is not expected to appear in the training data. Therefore, any small probability values less than 0.0006 can be chosen. A smaller probability value is better in magnifying the difference in the amount of the new initial state and new state transitions in attack activities than those resulting from noises in normal activities. In this study, we assign the probability value of 0.00001 to an initial state or a state transition, which does not appear in the training data, while using formula (5-6) to infer the probability of support to a sequence of states. This replacement of zero with a small probability value takes place during the inference and after the estimation of the Markov chain model from the training data is complete.

The learning and inference of the Markov chain model for attack detection are implemented using C++. In our C++ program, we use the data type "double" for the variable that keeps the probability of support to the sequence of states from a window. A variable of the data type "double" takes 64 bit of memory with the value range [1.0E-323, 1.0E+323] in scientific expression. For probability values smaller than 1.0E-323, the variable is assigned the value of zero. The results and discussion of this application are provided in Section V.

B. Application 2

A second application is presented in this section to further demonstrate the Markov chain technique using a larger dataset. In this situation we have two large datasets, a Mill dataset and a Pascal dataset, which were obtained from the MIT Lincoln laboratory. Mill and Pascal are the names of the host machines of the network constructed by the MIT Lincoln laboratory in order to simulate the environment of the network in the real world and thus provide a test bed of comprehensive evaluations for various intrusion detection systems. Our Mill and Pascal audit datasets are collected from these host machines. The operating systems are Solaris 2.5.1 and Solaris 2.7, respectively. The Mill dataset includes 15 normal sessions consisting of 68872 audit events and seven attack sessions. The Pascal dataset includes 63 normal sessions consisting of 81756 audit events and four attack sessions.

For our Markov chain technique, we use the data collected in the last hour from the machine named Mill, which only contained the normal audit data, as the training dataset. The dataset collected in all three hours from the same machine was used as the testing dataset. The data for the machine named Pascal is collected identically to that of the Mill machine. The results and discussion of this application are provided in Section V.

IV. TRAINING AND TESTING DATA

In this section, we present the training and testing data used in the example outlined in Section III.A involving four datasets with various levels of noise intrusion. Audit data of normal activities are required for estimating a Markov chain model of the norm profile. We use a sample of audit data for normal activities from the MIT (Massachusetts Institute of Technology) Lincoln Lab, containing a stream of 3019 audit events [3-12] for our study. Computer audit data for normal activities from the MIT Lincoln Lab are generated by simulating activities in a real computer and network system. This sample of audit data for normal activities, consisting of 2316 audit events, is divided into two parts; one part consisting of 1613 audit events and the other part consisting of the remaining 703 audit events. The part consisting of 1613 audit events is used for training a Markov chain model of the norm profile. The second part of the audit data, consisting of 703 audit events, is used for testing.

For testing, audit data of attack activities are generated in our lab by simulating 15 attack scenarios, in a random order, which have been collected over several years from various information sources. Some examples of the attack scenarios are password-guessing, attempts to gain an unauthorized access remotely, etc. The attack scenarios are manually run on the host machine while the auditing facility is turned on, resulting in a stream of 1225 audit events. As a result, the training data consist of the 1613 audit events for normal activities from the MIT Lincoln Lab, and the testing data consisting of the 703 audit events for normal activities from the MIT Lincoln lab and the 1225 audit events from attack activities from the simulation in our lab. Although there are two sources for obtaining data – MIT Lincoln Lab and our lab – the same operating system is used at both labs. The characteristic under study, event type, is also the same for both locations.

Next, a Markov chain model of the norm profile is obtained using the training data that contain the 1613 audit events for normal activities. As will be discussed in the next section, there are 284 possible types of audit events, of which only 11 types of audit events will appear in the training data. With 11 types of audit events, there are at most 132 parameters (equal to 121 parameters for the transition probability matrix + 11 parameters for the initial probability distribution) in the Markov chain model to estimate from the 1613 data points. Moreover, not all the 121 possible state transitions appear in the training data. The 1613 data points are sufficient to estimate the parameters in the Markov chain model in this situation.

For testing, the 1225 audit events (t = 0, 1, ..., 1224) from attack activities and the 703 audit events (t = 0, 1, ..., 702) from normal activities are viewed through a moving window of 100 events, creating 1126 windows (t = 99, 100, ..., 1224 or window no. 1-1126) for attack activities and 604 windows (t = 99, 100, ..., 702 or window no. 1-604) for normal activities.

Each of the first 99 audit events for either attack activities or normal activities does not create a window, because the event and preceding events together do not provide 100 audit events for a complete window. The robustness of the Markov chain technique will be examined for window sizes of $N = 10$ and $N = 100$ and the results compared and contrasted.

Note that intrusions usually occur in an information system while normal activities are also occurring in the information system. In real time, intrusive audit data are mixed with white noises of normal audit data. Thus, in order to investigate the robustness of the Markov chain technique or, how the performance of this technique depends on the quality of audit data, we create four sets of testing dataset using the 1225 audit events from attack activities and the 703 audit events from normal activities described above. Each subsequent dataset has an increase in mixture of abnormal audit events with the normal events (i.e., the noise level increases from one dataset to the next).

For testing dataset 1 (Pure), we put the 1225 abnormal audit events behind the 703 normal audit events. Obviously, we do not do any mixing for this set of data. For the testing dataset 2, the abnormal and normal audit events are mixed by dividing the 1225 abnormal audit events evenly into two parts, D_{a1} and D_{a2}, and also divide the 703 normal audit events evenly into two parts, D_{n1} and D_{n2}. We then arrange all parts as follows: D_{n1}, D_{a1}, D_{n2}, D_{a2}. We will refer to dataset 2 as the small noise level (SNL) dataset. For testing dataset 3, the 1225 abnormal audit events are divided into seven parts – not necessarily of the same size, D_{a1}, D_{a2}, D_{a3}, D_{a4}, D_{a5}, D_{a6} and D_{a7}, while the 703 normal audit events are also divided into seven parts, D_{n1}, D_{n2}, D_{n3}, D_{n4}, D_{n5}, D_{n6} and D_{n7}. We then arrange all these parts as follows: D_{n1}, D_{a1}, D_{n2}, D_{a2}, D_{n3}, D_{a3}, D_{n4}, D_{a4}, D_{n5}, D_{a5}, D_{n6}, D_{a6}, D_{n7}, D_{a7}. Dataset 3 will be referred to as the large noise level (LNL) dataset. For the testing dataset 4 (Random), we mixed the 1225 abnormal audit events and 526 normal audit events randomly. From a window at time *t*, a sequence of states $X_{t-99},\ldots, X_t$ is obtained and evaluated against the Markov chain model of the norm profile to yield the probability that the sequence of states is supported by this Markov chain model. A high probability indicates that the sequence of states more likely result from normal activities. The lower the probability we obtain, the less likely the sequence of states is a result of normal activities.

V. RESULTS AND DISCUSSION

A. Receiver Operating Characteristic (ROC) Curves

For a given test, different signal thresholds lead to different pairs of false alarm rate and hit rate that describe the performance of the test according to signal detection theory [5-55]. A false alarm would occur when normal activities were signaled as attack activities, and a hit would occur when a signal is observed and the event was an attack activity. The false-alarm rate is given by (number of signals)/(the number of normal activities). Obviously, we prefer that the false-alarm rate be small. For example, assume there are 703 normal audit events investigated and 24 of these events signaled as attack activities. We would say that the false-alarm rate is $\frac{24}{703}$ or 0.0341. Obviously, it is desirable to have the false-alarm rate be as small as possible. The hit rate, on the other hand, is given by (number of signals)/(number of attack activities). In this situation, we would prefer that the hit rate be as large as possible.

A Receiver Operating Characteristic (ROC) curve plots pairs of false-alarm rate and hit rate as points when various signal thresholds are used. For reasons discussed previously, we apply the logarithmic transformation given in (5-7) before the ROC curve is actually constructed.

For Application 2, event-based and session-based ROC curves are investigated. Session-based signal detection involves grouping a set of events into one "session" and estimating the average number of signals along with the standard deviation of the number of signals. For session-based detection, the signal threshold is initially estimated using $\overline{x}+3s$, where $\overline{x}$ is the average response and s represents the standard deviation of the response over the entire session. Based on normal theory, if the data are normally distributed we should expect approximately 99.7% of the data to fall within three standard deviations of the sample average. If data fall beyond three standard deviations, then we may conclude that these data are unusual (i.e., the result of attack activities). As will be shown, it is not necessary for three standard deviations to be used in calculating the signal threshold. We will provide some results when two standard deviations are employed for estimating the signal threshold.

To illustrate the usefulness of ROC curves in general, consider Figure 5-1. The closer the ROC curve is to the upper-left corner (representing 100% hit rate and 0% false alarm rate), the better the performance of the test. If the probability of support to the testing data from a moving window is greater than the decision threshold, the activities in the moving window are classified as normal activities; otherwise the activities in the moving window are classified as attack activities.

B. Application 1

ROC curves for window sizes of N = 100 and N = 10 are presented in Figure 5-1 and Figure 5-2, respectively. The results for each of the four small datasets described in the previous section are plotted for these window sizes.

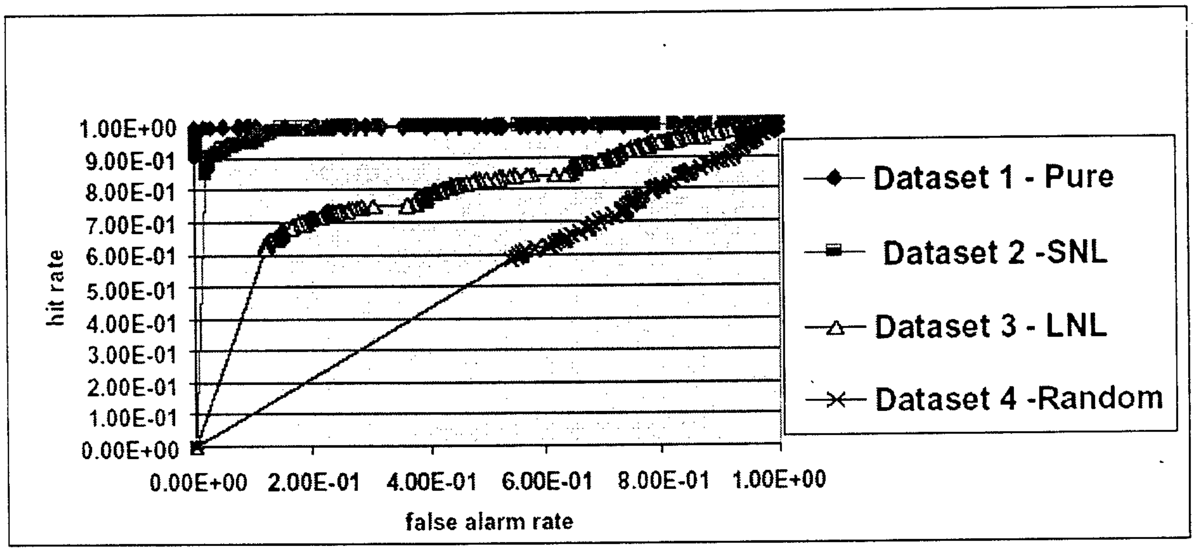


Figure 1. ROC Results of the Markov Chain Technique for Four Small Datasets and N = 100.

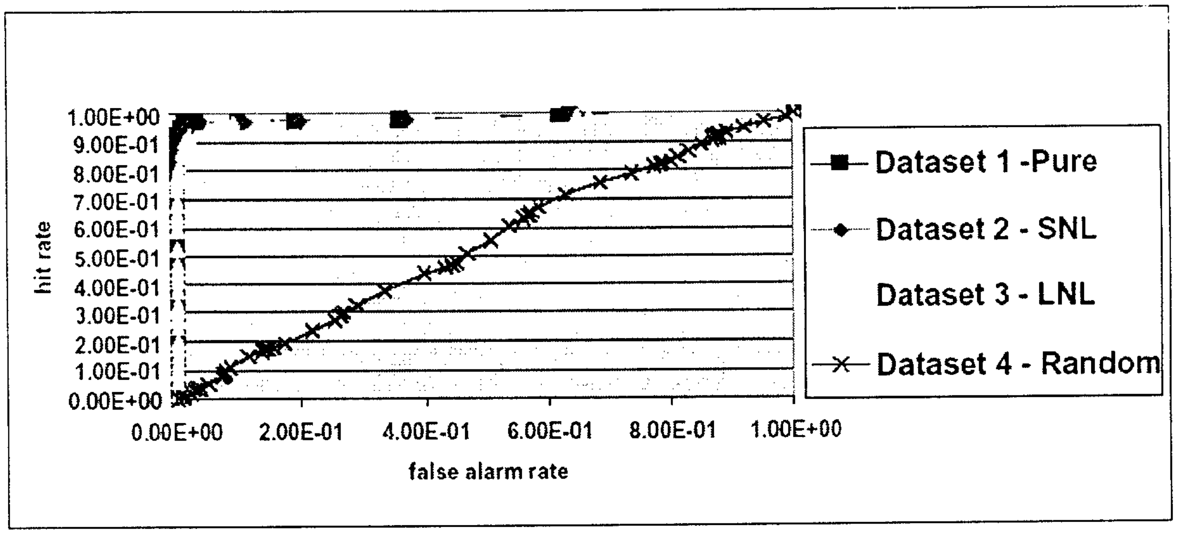


Figure 2. ROC Results of the Markov Chain Technique for Four Small Datasets and N = 10.

Examining the ROC curves in Figure 5-1 and Figure 5-2, it is obvious that the detection performance of the Markov chain technique depends strongly on the quality of audit data. That is, as the noise level increases the performance of the Markov chain decreases. We can also conclude that the Markov chain technique with a smaller window size (N = 10) outperforms the Markov chain technique with a window size of N = 100 as the amount of noise in the system increases.

C. Application 2

The ROC curves for the Mill dataset are given in Figure 5-3 through Figure 5-5. Figure 5-3 represents the Event-Based ROC curves for N = 100 and N = 10. Figures 5-4 and 5-5 display the Session-Based ROC curves using two standard deviations and three standard deviations, respectively. Based on these results, the detection performances of the Markov chain techniques with N = 100 and N = 10 are comparable for the event-based situation and the session-based situation with two standard deviations. Notice however, that as the standard deviation increases from two to three (see Figure 5-5) the Markov chain technique with N = 10 is superior to the technique when N = 100. The superior performance of the technique with a smaller window size is due in part to fewer opportunities for false alarms. A second observation is that the performance of Markov chain technique with N = 10 decreases as the number of standard deviations increase.

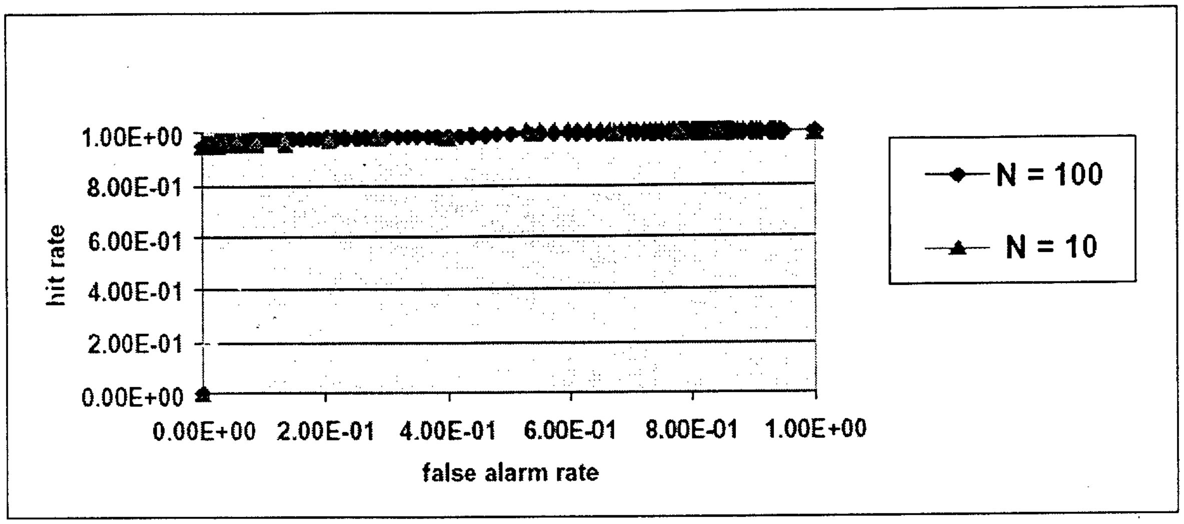


Figure 3. Event-Based ROC Curves for the Mill Dataset

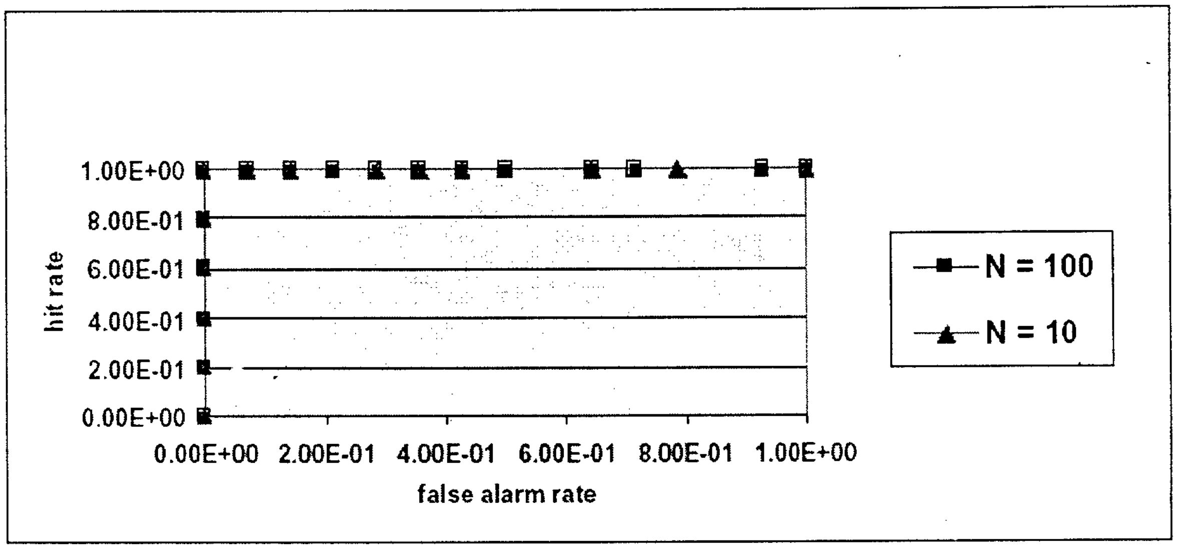


Figure 4. Session-Based ROC Curves for the Mill Dataset - Two Standard Deviations.

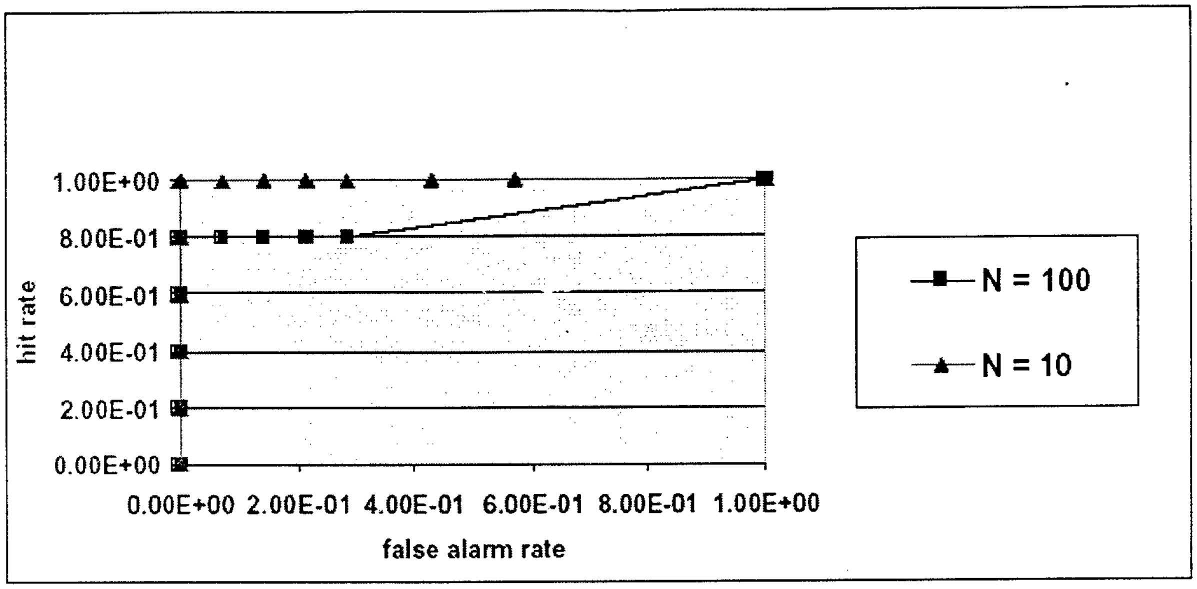


Figure 5. Session-based ROC curves for the Mill Dataset - Three Standard Deviations.

ROC curves are also provided for the Pascal dataset under the same conditions as the Mill dataset. The curves are provided in Figures 5-6 through 5-8. Based on these curves, the Markov chain technique appears to be quite robust to the window size in all three cases and to the number of standard deviations for the session-based cases shown in Figures 5-7 and 5-8. However, note that the false alarm rate is higher when N = 100 than when N = 10 for the session-based situations.

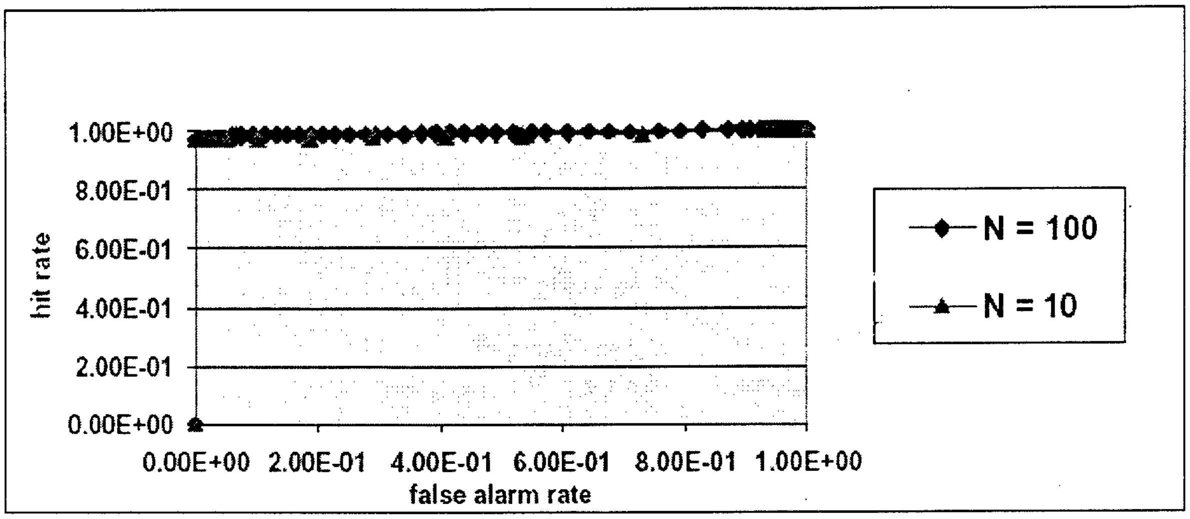


Figure 6. Event-Based ROC Curves for the Pascal Dataset.

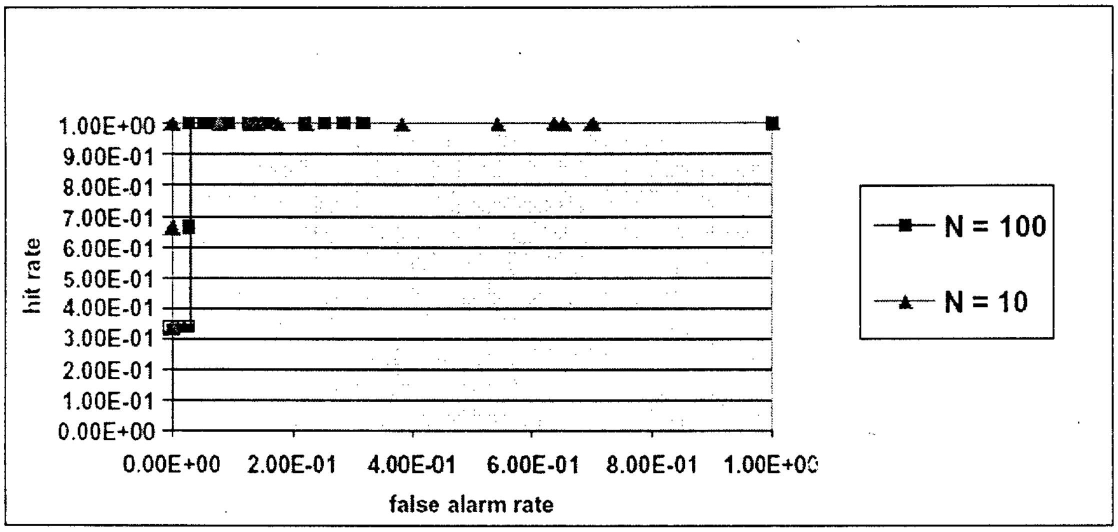


Figure 7. Session-Based ROC Curves for the Pascal Dataset - Two Standard Deviations.

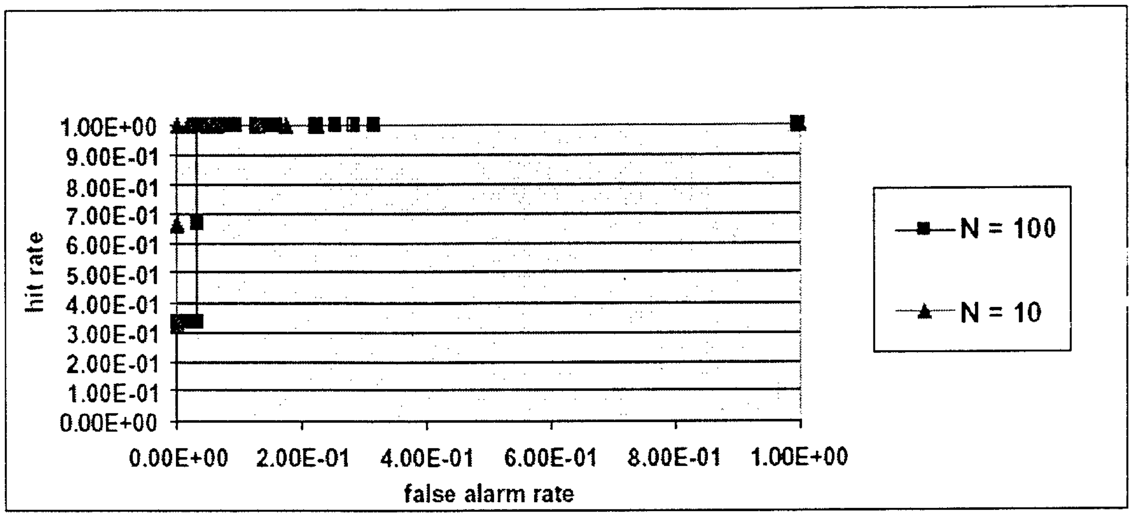


Figure 8. Session-Based ROC Curves for the Pascal Dataset - Three Standard Deviations.

VI. CONCLUSIONS

We conclude that in order to apply the Markov chain technique and other stochastic process techniques to modeling the sequential ordering of events, the quality of activity data needs to be improved. The improvement could be through, for example, noise canceling, if we consider data of normal activities are noises and data of intrusive activities are signals.

Our study has shown that the performance of the Markov chain techniques is not always robust to the window size. As the window size increases, the number of false alarms will also generally increase. Further investigation would reveal if the inclusion of noise cancellation techniques would improve the robustness of the Markov chain technique in relation to the window size.

We have also provided some initial results on the choice of the number of standard deviations when using session-based techniques. Our results indicate that as the number of standard deviations is increased, the poorer the performance of the Markov chain technique.

This should not be surprising since increasing the threshold would increase the probability of obtaining a false alarm.

Overall, our study provides some support for the idea that the Markov chain technique may not be as robust as the other intrusion detection methods such as the chi-square technique discussed in [5-35]. But, if the Markov chain approach is employed in conjunction with noise cancellation techniques, it is suspected that anomaly detection can be greatly improved. Future research will focus on improving the robustness of our Markov chain techniques to noise in the testing audit data.

REFERENCES

[5-1] W. Stallings. *Network and Inter-network Security Principles and Practice.* Englewood Cliffs, NJ: Prentice Hall, 1995.

[5-2] C. Kaufman, R. Perlman, and M. Speciner. *Network Security: Private Communication in a Public World.* Englewood Cliffs, New Jersey: Prentice Hall, 1995.

[5-3] T. Escamilla. *Intrusion Detection: Network Security beyond the Firewall.* New York: John Wiley & Sons, 1998.

[5-4] B. Simons. "Building big brother," Communications of the ACM, 43(1), pp. 31-32. January 2000.

[5-5] P. G. Neumann. "Risks of insiders," Communications of the ACM, 42(12), pp. 160, December 1999.

[5-6] M. Godwin. "Net to worry," Communications of the ACM, 42(12), pp. 15-17, December 1999.

[5-7] S. Jajodia, P. Ammann, and C. D. McCollum. "Surviving information warfare attacks," Computer, 32(4), pp. 57-63, April 1999.

[5-8] P. Mann. "Pentagon confronts mounting cyber risks," Aviation Week and Space Technology, 150(12), pp. 82-83, 22 March 1999.

[5-9] B. H. Barnes. "Computer security research: A British perspective," IEEE Software, 15(5), pp. 30-33, September/October 1998.

[5-10] A. Boulanger. "Catapults and grappling hooks: The tools and techniques of information warfare," IBM Systems Journal, 37(1), pp. 106-114, 1998.

[5-11] H. Debar, M. Dacier, and A. Wespi. "Towards a taxonomy of intrusion-detection systems," Computer Networks, 31, pp. 805-822, 1999.

[5-12] R. Lippmann, D. Fried, I. Graf, J. Haines, K., Kendall, D. McClung, D. Weber, S. Webster, D. Wyschogrod, R. Cunningham, and M. Zissman. "Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation." *In Proceedings of the DARPA Information Survivability Conference and Exposition.* Los Alamitos, CA. IEE Computer Society, pp. 12-26, January, 2000.

[5-13] D. Schnackenberg, K. Djahandari. "Infrastructure for intrusion detection and response," *In Proceedings of the DARPA Information Survivability Conference and Exposition.* Los Alamitos, CA: IEE Computer Society, pp. 3-11, January, 2000.

[5-14] T. Bass. "Intrusion detection systems and multi-sensor data fusion," Communications of the ACM, 43(4), pp. 99-105, April 2000.

[5-15] M. Stillerman, C. Marceau, and M. Stillman. "Intrusion detection for distributed applications," Communications of the ACM, 42(7), pp. 62-69, July 1999.

[5-16] U. Lindqvist, and P. A. Porras. "Detecting computer and network misuse through the production-based expert system toolset (P-BEST)," In Proceedings of the 1999 IEEE Symposium on Security and Privacy, IEEE, Oakland, CA, May 1999.

[5-17] P. A. Porras, and P. G. Neumann. "EMERALD: Event monitoring enabling responses to anomalous live disturbances," In Proceedings of NISSC, October 1997.

[5-18] P. G. Neumann, and P. A. Porras. "Experience with EMERALD to date," In Proceedings of the 1st USENIX Workshop on Intrusion Detection and Network Monitoring, Santa Clara, California, April 1999, pp. 73-80.

[5-19] W. Lee, and S. J. Stolfo. "Data mining approaches for intrusion detection," In Proceedings of the 7th USENIX Security Symposium, San Antonio, Texas, January 1998.

[5-20] W. Lee, S. J. Stolfo, and K. W. Mok. "A data mining framework for building intrusion detection models," In Proceedings of the 1999 IEEE Symposium on Security & Privacy, May 1999.

[5-21] W. Lee, S. J. Stolfo, and K. W. Mok. "Mining audit data to build intrusion detection models," In Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining, New York, NY, August 1998.

[5-22] W. Lee, S. J. Stolfo, and K. W. Mok. "Mining in a data-flow environment: Experience in network intrusion detection," In Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '99), San Diego, August 1999.

[5-23] G. Vigna, and R. Kemmerer. "NetStat: A network-based intrusion detection appoach." In Proceedings of the 14th Annual Computer Security Applications Conference, Scottsdale, Arizona, December 1998, http://www.cs.ucsb.edu/~kemm/netstat.html/.

[5-24] G. Vigna, S. T. Eckmann, and R. A. Kemmerer. "The STAT tool suit," In *Proceedings of the DARPA Information Survivability Conference and Exposition.* Los Alamitos, CA: IEE Computer Society, pp. 46-55, January, 2000.

[5-25] S. Kumar. Classification and Detection of Computer Intrusions. Ph.D. Dissertation, Department of Computer Science, Purdue University, West Lafayette, Indiana, 1995.

[5-26] N. Ye, X. Li, and S. M. Emran. "Decision trees for signature recognition and state classification," In Proceedings of the IEEE SMC Information Assurance and Security Workshop, West Point, New York, June 2000.

[5-27] C. Ko, G. Fink, and K. Levitt. "Execution monitoring of security-critical programs in distributed systems: A specification-based approach." In *Proceedings of the 1997 IEEE Symposium on Security and Privacy,* pp. 134-144, 1997.

[5-28] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle. "GrIDS – A graph-based intrusion detection system for large networks," In Proceedings of the 19th National Information Systems Security Conference, October 1996.

[5-29] T. Bowen, D. Chee, M. Segal, R. Sekar, T. Shanbhag, and P. Uppuluri. "Building survivable systems: An integrated approach based on intrusion detection and damage containment," *In Proceedings of the DARPA Information Survivability Conference and Exposition, Volume II.* Los Alamitos, CA: IEE Computer Society, pp. 84-99, January, 2000.

[5-30] D. E. Denning. "An intrusion-detection model," IEEE Transactions on Software Engineering, SE-13(2), pp. 222-232, February 1987.

[5-31] D. Anderson, T. Frivold, and A. Valdes. *Next-generation Intrusion Detection Expert System (NIDES): A Summary*. Technical Report SRI-CSL-97-07. Menlo Park, CA: SRI International, May, 1995.

[5-32] H. S. Javitz, and A. Valdes. "The SRI statistical anomaly detector." In *Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy*. May 1991.

[5-33] H. S. Javitz, and A. Valdes. *The NIDES Statistical Component Description of Justification*. Technical Report A010. Menlo Park, CA: SRI International, March, 1994.

[5-34] Y. Jou, F. Gong, C. Sargor, X. Wu, S. Wu, H. Chang, and F. Wang. "Design and implementation of a scalable intrusion detection system for the protection of network infrastructure." In *Proceedings of the DARPA Information Survivability Conference and Exposition*. Los Alamitos, CA: IEE Computer Society, pp. 69-83, 2000.

[5-35] N. Ye, Q. Chen, and S. M. Emran. "Chi-square statistical profiling for anomaly detection," In Proceedings of the IEEE SMC Information Assurance and Security Workshop, West Point, New York, June 2000.

[5-36] N. Ye, Q. Chen, and S. M. Emran. "Hotelling's T2 multivariate profiling for anomaly detection," In Proceedings of the IEEE SMC Information Assurance and Security Workshop, West Point, New York, June 2000.

[5-37] H. Debar, M. Becker, D. Siboni. "A neural network component for an intrusion detection system," In Proceedings of the 1992 IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, CA, May 1992, pp. 240-250.

[5-38] A. K. Ghosh, A. Schwatzbard, and M. Shatz. "Learning program behavior profiles for intrusion detection." In *Proceedings of the 1st USENIX Workshop on Intrusion Detection*

and Network Monitoring, Santa Clara, California, April, 1999, http://www.rstcorp.com/~anup/.

[5-39] S. Forrest, S. A. Hofmeyr, and A. Somayaji. "Computer immunology." *Communications of the ACM,* 40(10), pp. 88-96, October, 1997.

[5-40] H. Debar, M. Dacier, M. Nassehi, and A. Wespi. "Fixed vs. variable-length patterns for detecting suspicious process behavior," In Proceedings of the 5th European Symposium on research in Computer Security, Louvain-la-Neuve, Belgium, September 16-18, 1998, pp. 1-15.

[5-41] C. Warrender, S. Forrest, and B. Pearlmutter. "Detecting intrusions using system calls: Alternative data models," In Proceedings of the 1999 IEEE Symposium on Security and Privacy, pp. 133-145.

[5-42] W. DuMouchel. "Computer intrusion detection based on Bayes factors for comparing command transition probabilities," National Institute of Statistical Sciences, Technical Report No. 91, http://www.niss.org/downloadabletechreports.html.

[5-43] W.-H. Ju, and Y. Vardi. "A hybrid high-order Markov chain model for computer intrusion detection," National Institute of Statistical Sciences, Technical Report No. 92, http://www.niss.org/downloadabletechreports.html.

[5-44] M. Schonlau, W. DuMouchel, W.-H. Ju, A. F. Karr, M. Theus, and Y. Vardi. "Computer intrusion: Detecting masquerades," National Institute of Statistical Sciences, Technical Report No. 95, http://www.niss.org/downloadabletechreports.html.

[5-45] S. L. Scott. "Detecting network intrusion using a Markov modulated nonhomogeneous Poisson process," http://www-rcf.usc.edu/~sls/fraud.ps.

[5-46] N. Ye, Q. Zhong, and M. Xu. "Probabilistic networks with undirected links for anomaly detection," In Proceedings of the IEEE SMC Information Assurance and Security Workshop, West Point, New York, June 2000.

[5-47] W. DuMouchel, M. Schonlau. "A comparison of test statistics for computer intrusion detection based on principal components regression of transition probabilities," In Proceedings of the 30th Symposium on the Interface: Computing Science and Statistics.

[5-48] W. L. Winston, *Operations Research: Applications and Algorithms.* Belmont, CA: Duxbury Press, 1994.

[5-49] P. Buttorp. Stochastic Modeling of Scientific Data. London: Chapman & Hall, 1995.

[5-50] Gelman, Carlin, Stern, and Rubin. Bayesian Data Analysis. Chapman & Hall, 1995.

[5-51] I. L. MacDonald, and W. Zucchini. Hidden Markov and Other Models for Discrete Valued Time Series. London: Chapman & Hall, 1997.

[5-52] T. M. Mitchell, *Machine Learning.* Boston, MA: McGraw-Hill, 1997.

[5-53] F. V. Jensen, An Introduction to Bayesian Networks. London: UCL Press, 1996.

[5-54] N. Ye, and S. Vilbert. "The EWMA-EWMV technique for detecting anomalies in information systems." *IEEE Transactions on Reliability,* in review.

[5-55] B. H. Kantowitz and R. D. Sorkin. *Human Factors: Understanding People-System Relationships.* New York: John Wiley & Sons, 1983.

Chapter 6 First-Order versus High-Order Stochastic Models for Intrusion Detection

This chapter presents two different methods of applying stochastic models to computer intrusion detection. One method is based on a first-order stochastic model, specifically a Markov chain model. The other method is based on a partial high-order stochastic model. Stochastic models are used to build a profile of normal activities on a computer from training data of normal activities on the computer. The norm profile is then used to detect anomalous activities from testing data of both normal and intrusive activities on the computer for intrusion detection. Audit data of computer activities contain a sequence of computer events that is represented as a series of event transitions in stochastic models. The comparison of detection performance between the Markov chain model application and the partial high-order stochastic model application reveals better detection performance of the Markov chain model application to computer intrusion detection.

I. INTRODUCTION

A computer and network system consist of host machines (e.g., computers and routers) and communication links connecting those host machines. Intrusions into a computer and network system compromise the security of the system by making the system unavailable to provide services to users, corrupting data on systems, and/or stealing sensitive information on the system. Computer intrusion detection is to monitor activities on a computer or network system and detect intrusive activities that are taking place on the system [6-1 – 6-3]. Existing intrusion detection techniques fall in two general categories: signature recognition and anomaly detection

[6-4 – 6-7]. Signature recognition techniques acquire signature patterns of known intrusions and search for those intrusion signatures in data of system activities to detect intrusions. Anomaly detection techniques build a profile of normal system activities and consider any deviations of system activities from the norm profile as intrusions. This paper focuses on anomaly detection techniques for intrusion detection.

Two kinds of data have mainly been used for intrusion detection: computer audit data and network traffic data. Computer audit data record a sequence of events occurring on a host machine, while network traffic data capture data packets traveling on the network links connecting host machines. This study focuses on computer audit data.

Various methods have been used in existing intrusion detection techniques to represent a sequence of events in computer audit data for building the norm profile [6-5]. Different features of a given event sequence, such as the event frequency distribution and the sequential order of events, are captured in different methods of building and representing the norm profile. The event transition method captures the sequential order of events that is not available in the event frequency distribution method. The study by Ye and her colleagues [6-5] revealed that the sequential order of events provided additional advantages to the event frequency distribution for intrusion detection performance. That is, the sequential order of events is important to intrusion detection, because intrusive events follow a certain logical sequence in order to produce the intrusion effect.

Both first-order and high-order models of stochastic process have been applied for capturing the sequential order of events in terms of event transitions for intrusion detection [6-5, 6-8 – 6-16]. Theoretically high-order stochastic models capture more information of a given event sequence than first-order stochastic models. However, high-order stochastic models come at high costs of computation. Hence, there are studies that use artificial neural networks [6-13 – 6-14] and other techniques [6-15 – 6-16] for prediction of the next event, E_n, from *(T-1)* previous events, E_{n-1}, …, E_{n-T+1}. The model of the transition from *k* previous events to the next event in those techniques is actually a partial high-order stochastic model, because the model does not include information about the transition of *(T-2)* previous events to the next events, the transition of *(T-3)* previous events to the next events, and so on. Although building a partial high-order stochastic model for intrusion detection reduces the computational cost, it is not clear whether the partial high-order stochastic model can still produce the same level of detection performance as the complete high-order stochastic model.

This study investigates the performance of a partial high-order stochastic model in comparison with the performance of a first-order stochastic model—a Markov chain model—for intrusion detection. The performance of a partial high-order stochastic model is compared not with that of a complete high-order stochastic model but with that of a first-order stochastic model due to the cost of building a complete high-order stochastic model.

In the following sections of this chapter, we first describe the first-order Markov model and the partial high-order Markov model used for intrusion detection. We then describe the application of these Markov models to intrusion detection, including data sources and problem representation. Finally, we present the testing results of these applications, and conclude the paper.

II. FIRST-ORDER AND HIGH-ORDER STOCHASTIC MODELS

If we observe a system at some discrete points in time, $n = 0, 1, 2, 3, 4, \ldots$, the state of the system is defined by a random variable, X_n, and the value of X_n is not known with certainty before time n, then $\{X_n, n \geq 0\}$ describes a stochastic process of the system [5-17]. This type of stochastic process, in which the system is observed at discrete points in time, is known as *discrete-time stochastic process*. The times at which the system is observed may not be equally spaced along the time axis. The other form of stochastic process is *continuous-time stochastic process*, in which a system is observed continuously over time. We are interested in discrete-time stochastic processes in this study. The first-order and high-order models of discrete-time stochastic process are described below.

A. A Markov Chain Model—A First-Order Stochastic Model

A Markov chain is a first-order discrete-time stochastic process with the Markov property as follows:

$$P(X_{n+1} = i_{n+1} \mid X_n = i_n, X_{n-1} = i_{n-1}, \ldots, X_0 = i_0) = P(X_{n+1} = i_{n+1} \mid X_n = i_n) \qquad (6\text{-}1)$$

for all n, and $i_0, i_1, \ldots, i_{n+1}$ are system states [6-18]. That is, the probability distribution of the state at time $n+1$ depends on the state at time n, and does not depend on the previous states leading to the state at time n. A Markov chain model is a first-order stochastic model.

Adding an additional property that a state transition from time n to time $n+1$ is independent of time:

$$P(X_{n+1} = i_{n+1} \mid X_t = i_n) = P(X_{t+1} = j \mid X_t = i) = p_{ij} \tag{6-2}$$

for all n and all states, the Markov chain becomes the stationary Markov chain [6-24] which we refer to in later text simply as the Markov chain, where p_{ij} is the probability that the system is in a state j at one time given that the system is in state i at the previous time [6-18]. If the system has a finite number of states, 1, 2, ..., s, the Markov chain model can be defined by a transition probability matrix [6-18]:

$$P = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1s} \\ p_{21} & p_{22} & \cdots & p_{2s} \\ \vdots & \vdots & \vdots & \vdots \\ p_{s1} & p_{s2} & \cdots & p_{ss} \end{bmatrix}, \tag{6-3}$$

and an initial probability distribution [6-18]:

$$Q = [q_1 \quad q_2 \quad \cdots \quad q_s], \tag{6-4}$$

where q_i is the probability that the system is in state i at time 0, and

$$\sum_{j=1}^{j=s} p_{ij} = 1. \tag{6-5}$$

The joint probability for a given sequence of T states X_{n-T+1}, ..., X_n at time $n-(T-1)$, ..., n is computed as follows:

$$P(X_{n-T+1}, \cdots, X_n) = q_{x_{n-T+1}} \prod_{t=T-1}^{1} p_{X_{n-t}X_{n-t+1}} \tag{6-6}$$

The transition probability matrix and the initial probability distribution of a Markov chain model can be learned from observations of system state in the past. Provided with observations of system state, X_0, X_1, X_2, ..., X_{N-1} at time n = 0, ..., $N-1$, we learn the transition probability matrix and the initial probability distribution as follows [6-19 – 6-20]:

$$p_{ij} = \frac{N_{ij}}{N_{i.}} \tag{6-7}$$

$$q_i = \frac{N_i}{N} \tag{6-8}$$

where

N_{ij} is the number of observation pairs X_n and X_{n+1} with X_n in state i and X_{n+1} in state j,

$N_{i.}$ is the number of observation pairs X_n and X_{n+1} with X_n in state i and X_{n+1} in any one of the states 1, …, s,

Ni is the number of X_n's in state i, and

N is the total number of observations.

Bayes parameter estimation can also be used to learn the transition probability matrix and the initial probability distribution from historic data. However, Bayes parameter estimation has a high computational cost and is not appropriate when there are large amounts of data. Provided with sufficient historic data, the estimation of the transition probability matrix and the initial probability distribution via formulas (7) and (8) can be quite stable.

B. A Partial High-Order Stochastic Model

Given a discrete-time stochastic process and a sequence of states from this process, X_0, X_1, …, X_n, the joint probability for the sequence of states is calculated as follows:

$$P(X_n, X_{n-1}, \dots, X_0) = P(X_n \mid X_{n-1}, \dots, X_0) P(X_{n-1} \mid X_{n-2}, \dots, X_0) \dots P(X_2 \mid X_1) P(X_0). \tag{6-9}$$

If we assume the state at time n depends only on previous $(T-1)$ states, the joint probability for a given sequence of states, X_{n-T+1}, …, X_n is calculated as follows:

$$P(X_{n-T+1}, \dots, X_{n-1} X_n) = P(X_n \mid X_{n-1}, \dots, X_{n-T+1}) P(X_{n-1} \mid X_{n-2}, \dots, X_{n-T+1}) \dots P(X_{n-T+1}) \tag{6-10}$$

Hence, a complete high-order stochastic model should include the following transition matrices:

$$P(X_n \mid X_{n-1},\dots,X_{n-T+1}),$$

$$P(X_{n-1} \mid X_{n-2},\dots,X_{n-T+1}),$$

$$\dots$$

$$P(X_{n-T+2} \mid X_{n-T+1}), \qquad (6\text{-}11)$$

and the initial probability distribution:

$$Q = [q_1 \quad q_2 \quad \cdots \quad q_s], \qquad (6\text{-}12)$$

where q_i is the probability that the system is in state i at time 0, and $i \in S$.

In those studies [6-13 – 6-16] based on the prediction of the next event, X_n, from $(T-1)$ previous events, X_{n-T+1}, ..., X_{n-1} for intrusion detection, training data of normal activities on a computer or network system are used to build a high-order stochastic model of normal activities (norm profile) with only the information of

$$P(X_n \mid X_{n-1},\dots,X_{n-T+1}) \qquad (6\text{-}13)$$

which is represented in the form of an artificial neural network (ANN) or simply a look-up table. Given a sequence of events in the testing data, X_{n-T+1}, ..., X_n at time $n-T+1$, ..., n, the same term in formula (6-13), $P(X_n \mid X_{n-1} X_{n-2} \dots X_{n-T+1})$, is determined from the norm profile through making a prediction using the ANN model or the look-up table. If this probability, $P(X_n \mid X_{n-1} E_{n-2} \dots X_{n-T+1})$, is smaller than a pre-set threshold, the sequence of events is considered as intrusive. That is, in those studies only a small part of formulas (6-10) and (6-11) is used for intrusion detection, whereas a complete high-order stochastic model consists of formulas (6-10), (6-11) and (6-12) in their complete form. Hence, only a partial high-order stochastic model is used for the application to intrusion detection.

III. APPLICATION OF STOCHASTIC MODELS TO INTRUSION DETECTION

In this section, we describe the data of system activities that are used for intrusion detection. We also define how the Markov chain model and the partial high-order stochastic model are applied to the problem of intrusion detection.

A. Training and Testing Data

A computer network consists of mainly host machines that are connected through communication links. Host machines may run operating systems such as UNIX or Windows NT/2000. These operating systems have facilities to monitor and log activities occurring on a host machine, producing compute audit data or log data. Network traffic data are usually collected over communication links while these data are traveling through communication links. In this study, we detect intrusions based on data of activities on host machines instead of network traffic data.

Activities on host machines generate sequences of computer actions that in turn induce sequences of auditable events in the kernel of operating systems recorded in form of computer audit data. Audit data contain records of auditable events, each audit record for each event.

Audit data from two UNIX-based host machines, named Mill and Pascal, are available in the 2000 DARPA (Defense Advanced Research Projects Agency) Evaluation Data that are generated by simulating activities in a real world computer and network system. Normal and intrusive activities are similar on Mill and Pascal. From these two host machines, we obtain two audit data sets (Mill and Pascal) for this study. The operating systems for the Mill and Pascal host machines are Solaris 2.5.1 and Solaris 2.7 respectively. On these operating systems, there is a Basic Security Module (BSM) that record audit events and produce audit data. Information in an audit record for an audit event includes event type, user ID, process ID, session ID, etc. There are 284 different types of auditable events in BSM. In this study, we extract only the information of event type from each audit record because event type has been very effective in detecting intrusive activities [6-5, 6-6].

Both Mill and Pascal data sets are used to test the intrusion detection performance of stochastic models, producing two sets of testing results respectively. Table 6-1 summarizes the main features of Mill/Pascal data sets. Mill/Pascal data cover about 3 hours of activities on Mill/Pascal host machines.

Table 6-1. Features of Mill and Pascal data sets.

Features	**Mill Data Set**	**Pascal Data Set**
Normal Events	68,871	81,755
Intrusive Events	36,036	32,327
All Events	104,907	114,082
Number of Unique Event Types	69	53
Number of Normal Sessions	14	63
Number of Intrusive Sessions	7	4

We use two sets of training and testing data: Mill and Pascal. Training data are required to build a stochastic model of the norm profile. From the Mill data set, audit data for normal events in the last hour of the 3-hour activities are used as the training data, and the whole set of the 3-hour Mill data containing both normal and intrusive activities is used as the testing data.

There are 42,983 audit events in the Mill set of training data with 67 unique event types. From the Pascal data set, audit data for normal events in the last hour of the 3-hour activities are used as the training data, and the whole set of the 3-hour Pascal data containing both normal and intrusive activities is used as the testing data. There are 20,616 audit events in the Pascal set of training data with 53 unique event types.

B. Data Representation

A sequence of audit events in the training or testing data presents a discrete-time stochastic process that we consider for intrusion detection. Each audit event is an observation, and the time when an audit event occurs is the time of the observation. We consider the event type of an audit event as the state. Hence, there are 284 possible states, denoted by *s1, ..., s284*, as there are 284 different types of audit events. A set of training or testing with N audit events can thus be represented as a sequence of states, $X_0, X_1, \ldots, X_{N-1}, X_N$.

C. Application of the Markov Chain Model to Intrusion Detection

Using a set of training data, a Markov chain model of the norm profile is built based on formulas (6-7) and (6-8). Each audit event in a set of testing data is evaluated to determine whether it is intrusive or not. To determine whether the event at time *n* is intrusive or X_n is intrusive, we use the state information from this audit event and previous audit events within a window of the recent past because the state information from a single event is not sufficient to detect intrusions [6-5]. A window of size T contains an audit event and previous T-1 events. Hence, a window of size T presents a sequence of T states, $X_{n-T+1}, \ldots, X_{n-1}, X_n$. Given a sequence of states in a recent-past window of the current event, we use formula (6-6) to obtain the joint probability for this sequence of states. This joint probability provides the basis to evaluate whether the current event is intrusive or not. The larger the value of this joint probability, the more likely the current event is normal as we more likely observe this sequence of states in the condition of normal activities that is captured in the Markov chain model of the norm profile.

There may be certain event types or transitions between certain event types that appear in a set of testing data but not in a set of training data, resulting in that values of certain terms in the right-hand side of formula (6-6) are not available in the transition probability matrix or the initial probability distribution of the Markov chain model. We assign a small value, specifically a default value of 0.000001, to those terms. By assigning a small value instead of the zero value to those terms, we avoid getting the zero value for the calculation of a joint probability. The non-zero value of a joint probability helps distinguish small effects of anomalies caused by noises in the data of normal activities from large effects of anomalies caused by intrusive activities.

In this study, we test the Markov chain model application using two window sizes of T=10 and T=100 for comparison and evaluation of possible effects of the window size on the detection performance.

D. Application of the High-Order Stochastic Model to Intrusion Detection

To build a partial high-order stochastic model of the norm profile from a set of training data, we use formula (6-13) to obtain a table (see Table 6-2) from the set of training data. This table includes the probabilities of the current state conditioned on the sequence of previous (T-1) states.

Table 6-2. The transition probability table in the partial high-order stochastic model.

Sequence of Previous T-1 States	Probability of the Current State, X_n			
	s1	……	s284	Total Count
$X_{n-T+1} \ldots X_{n-1}$	2/7		4/7	7
…				

Each audit event or state, X_n, in a set of testing data is evaluated to determine whether it is intrusive or not by examining a sequence of states in a recent-past window of size T, X_{n-T+1}, …, X_{n-1}, X_n. Formula (6-13) is used to get a probability of this sequence of states by looking up the transition probability table in the partial high-order stochastic model as shown in Table 6-2. This probability provides the basis to evaluate whether the current event is intrusive or not. The larger the value of this probability, the more likely the current event is normal as we more likely observe this sequence of states in the condition of normal activities that is captured in the partial high-order stochastic model of the norm profile. The small default value of 0.000001 is also assigned to any term in the right-hand side of formula (6-13) that is not available in the transition probability table.

In this study, we test the Markov chain model application using two window sizes of T=10 and T=100 for comparison and evaluation of possible performance differences caused by different window sizes.

IV. RESULTS AND DISCUSSIONS

In this section, we present and discuss the results of testing the Markov chain model application and the partial high-order stochastic model application.

A. Analysis of the Testing Results

We first define some terms that are used in the analysis of the testing results.

1. Hit rate: the rate at which intrusive events are correctly identified. It is the ratio of the number of correctly identified intrusive events to the total number of intrusive events in a set of testing data.

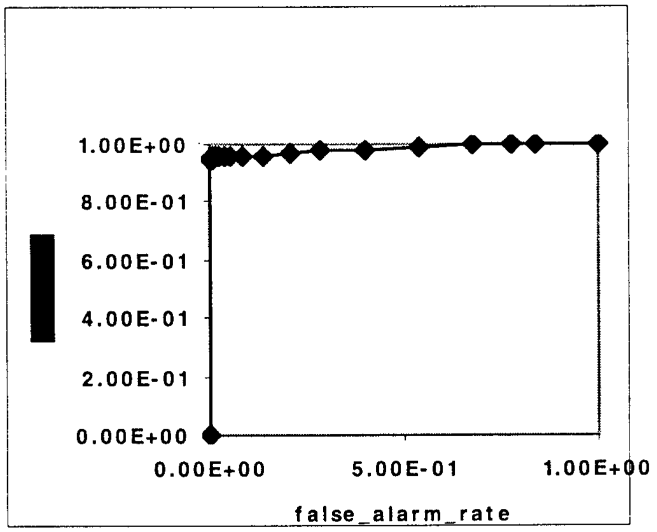
2. False alarm rate: the rate at which normal events are falsely identified as intrusive events. It is the ratio of the number of normal events identified as intrusive events to the total number of normal events in a set of testing data.

3. Signal threshold: a value chosen on the probability scale between 0.0 and 1.0. It's used as a point to distinguish intrusive events from normal events. For example, for a signal threshold point of 0.5 on a probability scale, we considered all events with probabilities at or above 0.5 as normal and those below 0.5 as attack activities.

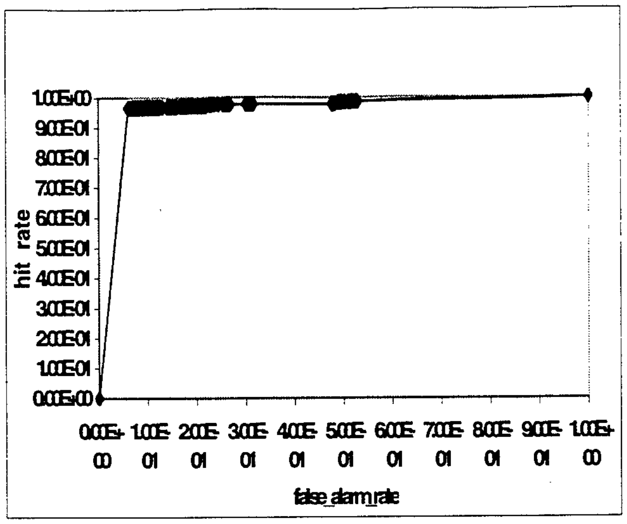
A good detection performance should be characterized by a high hit rate and a low false alarm rate. We use a Receiver Operating Characteristic (ROC) curve [6-21 – 6-22] to analyze the detection performance of the Markov chain model application and the partial high-order stochastic model application. Given a signal threshold, we can identify each audit event in a set of testing data as normal or intrusive based on the probability value from formula (6-6) for the Markov chain model application and from formula (6-13) for the partial high-order stochastic model application. After identifying each audit event in a set of testing data as normal or intrusive, we compute the hit rate and the false alarm rate. Hence, for a given signal threshold, we obtain a pair of hit rate and false alarm rate. By varying the value chosen for the signal threshold, we obtain pairs of hit rate and false alarm rate. A ROC curve plots those pairs of false alarm rate and hit rate as points on the curve. The upper-left corner of the chart in which a ROC curve is plotted represents the point of the 100% hit rate and the 0% false alarm rate. Hence, the closer a ROC curve is to the upper-left corner of the chart, the better the detection performance.

B. Testing Results on Mill Data Set

Figure 6-1 shows the ROC curves of the Markov chain model application and the partial high-order stochastic model using the window size of ten, T=10. Figure 6-2 shows the ROC curves of the Markov chain model application and the partial high-order stochastic model using the window size of one hundred, T=100. Regardless of different window sizes, the Markov chain model application produces better performance than the partial high-order stochastic model on Mill data set.

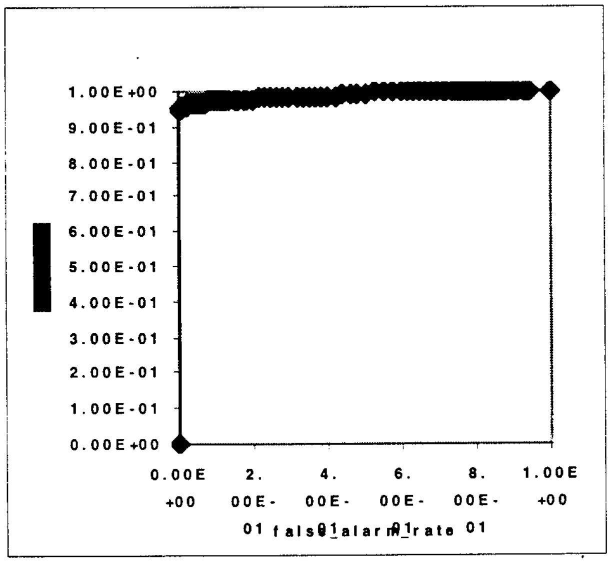


(a) The Markov chain model application.

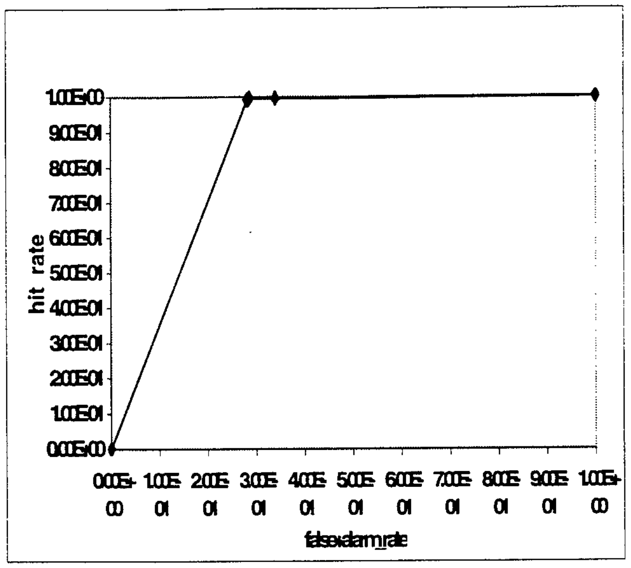


(b) The partial high-order stochastic model application.

Figure 6-1. The detection performance of the Markov chain model application and the partial high-order stochastic model application using window size of ten for Mill data set.



(a) The Markov chain model application.

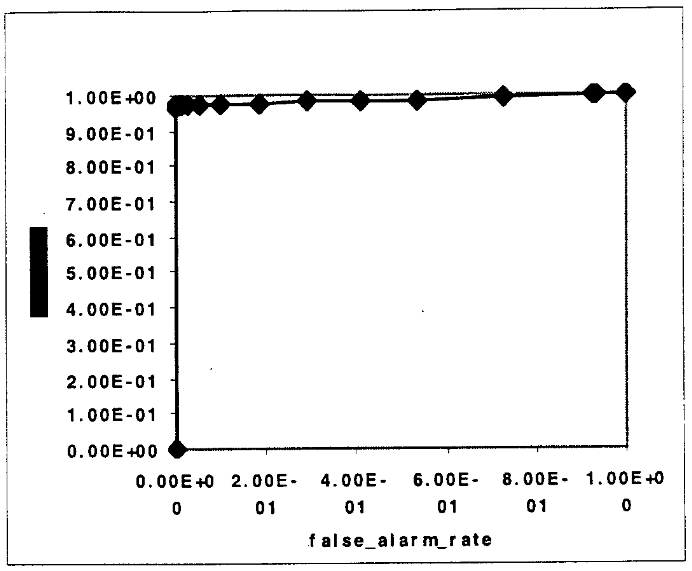


(b) The partial high-order stochastic model application.

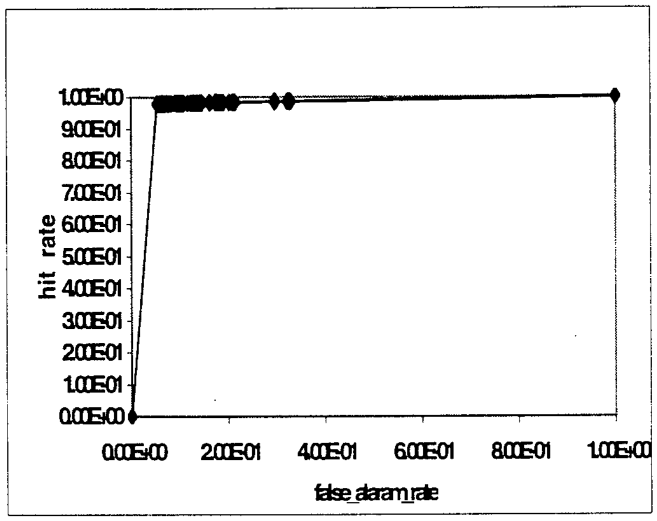
Figure 6-2. The detection performance of the Markov chain model application and the partial high-order stochastic model application using window size of one hundred for Mill data set.

C. Testing Results on Pascal Data Set

Figure 6-3 shows the ROC curves of the Markov chain model application and the partial high-order stochastic model using the window size of ten, T=10. Figure 6-4 shows the ROC curves of the Markov chain model application and the partial high-order stochastic model using the window size of one hundred, T=100. Regardless of different window sizes, the Markov chain model application produces better performance than the partial high-order stochastic model on Pascal data set.

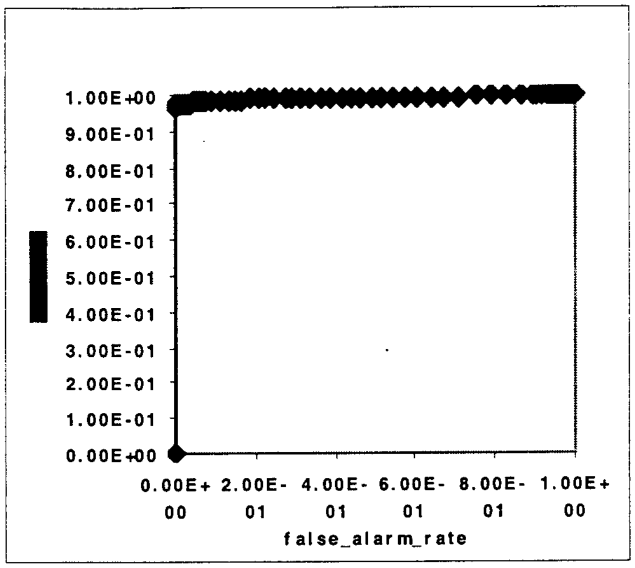


(a) The Markov chain model application.

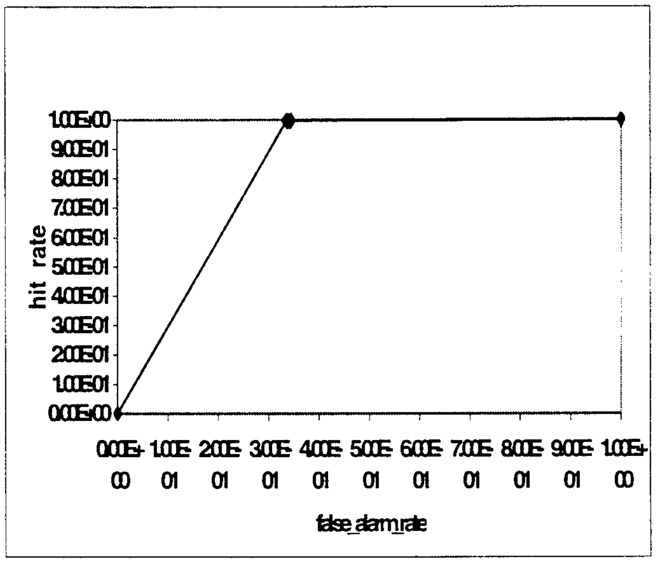


(b) The partial high-order stochastic model application.

Figure 6-3. The detection performance of the Markov chain model application and the partial high-order stochastic model application using window size of ten for Pascal data set.



(a) The Markov chain model application.



(b) The partial high-order stochastic model application.

Figure 6-4. The detection performance of the Markov chain model application and the partial high-order stochastic model application using window size of one hundred for Pascal data set.

D. Discussions of the Testing Results

The testing results in this study show that the Markov chain model application consistently produces better performance than the partial high-order stochastic model for intrusion detection, regardless of different window sizes (T=10 and 100) and different data sets (Mill and Pascal). This may be attributed to the incomplete high-order stochastic model built in training and the incomplete inference of the probability for a sequence of states made in testing. It is possible that performance of a high-order stochastic model application can be improved if a complete high-order stochastic model is built in training and is used in testing to compute the joint probability for a sequence of states. However, building such a complete high-order stochastic model is computationally expensive, and prevents its practical application. It is also a challenge to determine an appropriate order of a stochastic model application. Based on the testing results of this study, the first-order stochastic model, specifically the Markov chain model, appears to provide a viable solution to make a trade-off between performance and computation cost.

V. CONCLUSION

It has been popular in existing studies to represent the sequential order of events for intrusion detection based on fundamentally a partial high-order stochastic model and a single high-order event transition for prediction. However, we show in this study that the performance of such a partial high-order stochastic model application to intrusion detection is not as good as a complete Markov chain model application—a first-order stochastic model application—to intrusion detection. Considering the low computational cost of the Markov chain model application and its demonstrated performance as shown in this study, we recommend the use of the Markov chain model to capture the sequential order of computer or network events for intrusion detection.

REFERENCES

6-1. Escamilla T. *Intrusion Detection: Network Security beyond the Firewall.* John Wiley & Sons: New York, 1998.

6-2. Lippmann R, Fried D, Graf I, Haines J, Kendall K, McClung D, Weber D, Webster S, Wyschogrod D, Cunningham R, Zissman M. Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation. In *Proceedings of the DARPA Information Survivability Conference and Exposition.* IEEE Computer Society: Los Alamitos, CA: IEEE Computer Society, pp. 12-26, January 2000.

6-3. Debar H, Dacier M, Wespi A. Towards a taxonomy of intrusion-detection systems. *Computer Networks* 1999 **31**: 805-822.

6-4. Ye N, Giordano J, Feldman J. A process control approach to cyber attack detection. *Communications of the ACM* 2001 **44**(8): 76-82.

6-5. Ye N, Li X, Chen Q, Emran SM, Xu M. Probabilistic techniques for intrusion detection based on computer audit data. *IEEE Transactions on Systems, Man, and Cybernetics* 2001 **31**(4): 266-274.

6-6. Ye N, Chen Q. An anomaly detection technique based on a chi-square statistic for detecting intrusions into information systems. *Quality and Reliability Engineering International* 2001 **17**(2): 105 - 112.

6-7. Emran SM, Ye N. A system architecture for computer intrusion detection. *Information, Knowledge, Systems Management* 2001 **2**(3): 271-290.

6-8. Warrender C, Forrest S, Pearlmutter B. Detecting intrusions using system calls: Alternative data models. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, pp. 133-145, 1999.

6-9. DuMouchel W. *Computer Intrusion Detection Based on Bayes Factors for Comparing Command Transition Probabilities.* National Institute of Statistical Sciences, Technical Report No. 91, http://www.niss.org/downloadabletechreports.html.

6-10. Ju WH, Vardi Y. *A Hybrid High-order Markov Chain Model for Computer Intrusion Detection.* National Institute of Statistical Sciences, Technical Report No. 92, http://www.niss.org/downloadabletechreports.html.

6-11. Schonlau M, DuMouchel W, Ju WH, Karr AF, Theus M, Vardi Y. *Computer Intrusion: Detecting Masquerades.* National Institute of Statistical Sciences, Technical Report No. 95, http://www.niss.org/downloadabletechreports.html.

6-12. Scott SL. *Detecting Network Intrusion Using a Markov Modulated Nonhomogeneous Poisson Process.* http://www-rcf.usc.edu/~sls/fraud.ps.

6-13. Debar H, Becker M, Siboni D. A Neural Network Component for an Intrusion Detection System. In *Proceedings of the 1992 IEEE Computer Society Symposium on Research in Security and Privacy*, Oakland, CA, pp. 240-250, May 1992.

6-14. Ghosh AK, Schwatzbard A, Shatz M. Learning Program Behavior Profiles for Intrusion Detection. In *Proceedings of the 1st USENIX Workshop on Intrusion Detection and Network Monitoring,* Santa Clara, California, April, 1999, http://www.rstcorp.com/~anup/.

6-15. Wespi, A., H. Debar, M. Dacier and M. Nassehi. "Fixed- vs. Variable-Length Patterns for Detecting Suspicious Process Behavior." Journal of Computer Security. 8 (2000), IOS Press, 159-181.

6-16. Eskin E, Lee W, Stolfo SJ. Modeling system calls for intrusion detection with dynamic window sizes. In *Proceedings of the Second DARPA Information Survivability Conference and Exposition (DISCEX II)*, pp. 165-175, 2001.

6-17. Buttorp P. *Stochastic Modeling of Scientific Data.* Chapman & Hall: London, 1995.

6-18. Winston WL. *Operations Research: Applications and Algorithms.* Duxbury Press: Belmont, California, 1994.

6-19. Mitchell TM, *Machine Learning.* McGraw-Hill: Boston, MA, 1997.

6-20. Jensen FV. *An Introduction to Bayesian Networks.* UCL Press: London, 1996.

6-21. Swets JA. The Relative Operating Characteristic in Psychology. *Science* 1973 **182**: 990–1000.

6-22. Egan JP. *Signal detection theory and ROC-analysis.* Academic Press: New York, 1975.

Chapter 7 Grid- and Dummy-Cluster-Based Learning of Normal and Intrusive Clusters

Many existing signature recognition techniques for intrusion detection cannot handle huge amounts of complex data from computer and network systems to detect intrusions in a scalable, incremental manner. This chapter presents an application of an innovative data mining algorithm—CCAS—to intrusion detection through intrusion signature recognition. CCAS provides a scalable, incremental procedure to learn clusters of different classes (i.e., normal and intrusive classes) from historic training data of normal and intrusive activities in computer and network systems. These clusters of normal and intrusive computer activities are used to classify observed data of computer activities for intrusion detection. Two different methods of learning clusters are developed, tested and compared: grid-based and dummy-cluster-based. Training and testing data are computer audit data produced by the Basic Security Module of a Solaris operating system to record activities in a UNIX-based host machine. The two methods of CCAS are tested using four different input orders of training data points to examine the robustness (sensitivity) of these methods to the input order of training data points. The detection performance and robustness of both CCAS methods are analyzed. The testing results show that different input orders of training data have certain impact on performance of both methods. The impact on the performance of CCAS based on dummy clusters is more significant when all normal data are presented first before attack data in the training data set. CCAS based on dummy clusters produce better performance than grid-based CCAS for three of the four input orders, and in overall produce fewer clusters and thus require less computation time in clustering and classification for intrusion detection.

I. INTRODUCTION

Computer intrusion detection is an essential part of protecting computer and network systems from internal or external attacks (intrusions) that compromise security of computer and network systems. Existing intrusion detection systems use two types of activity data from a computer and network system: network traffic data and computer audit data. There are two general categories of intrusion detection techniques: anomaly detection and signature recognition (misuse detection) [7-1, 7-2]. Anomaly detection techniques first build a profile of a subject's normal activities and then signal observed activities as intrusive if they deviate significantly from the profile of normal activities. Signature recognition techniques first learn signature patterns of normal and intrusive activities from training data, and then match these signature patterns with incoming data of observed activities to determine whether the activities are normal or intrusive. Anomaly detection is capable of detecting novel attacks, while signature recognition is accurate in detecting known attacks. Anomaly detection and signature recognition often co-exist in an intrusion detection system to complement each other.

This chapter focuses on signature recognition techniques for intrusion detection. Many data mining techniques, such as decision trees [7-3], association rules [7-4], artificial neural networks and genetic algorithms [7-5, 7-6], and Bayesian networks [7-7], have been used for intrusion detection through signature recognition. Those data mining techniques are used to learn signature patterns of normal and intrusive activities from training data, and use those signature patterns to classify data of observed activities as normal or intrusive.

However, these existing data mining techniques are not capable of learning signature patterns from large amounts of activity data from computer and network systems in a scalable, incremental manner. First of all, activity data from a computer and network system can easily contain millions of records per day. In addition, each record may have hundreds of data fields. A data mining algorithm to learn signature patterns from such large amounts of data must be scalable. Secondly, patterns of normal and intrusive activities likely change over time, and new forms of attacks appear constantly. Hence, a data mining algorithm must have the incremental learning ability to update signature patterns as more training data of normal and intrusive activities become available.

We have developed an innovative data mining algorithm [7-8, 7-9], called Clustering and Classification Algorithm – Supervised (CCAS), which has the scalable, incremental learning ability. CCAS is based on two concepts: supervised clustering for learning signature patterns of normal and intrusive activities during the training phase, and instance-based classification for using signature patterns of normal and intrusive activities to classify observed activities during the testing phase.

While incrementally clustering data records in the training data set during the training phase, CCAS uses dummy clusters to prevent consecutive data records from being clustered together even though their attribute values are not similar. We call this dummy-cluster-based CCAS. This method of avoiding "local bias of input order" is different from the grid-based method found in some existing clustering techniques such as WaveCluster [7-10] and MAFIA [7-11]. Both dummy-cluster- and grid-based methods are heuristic methods. To investigate which method is better at avoiding "local bias" and thus is more robust or less sensitive to the order of training data records, we develop a grid-based CCAS, and conduct a study to compare detection performance and robustness of the two CCAS methods. This paper presents the results from this study.

In this chapter, clustering techniques and instance-based classification techniques are reviewed first to lay the theoretical foundation for concepts used in CCAS. Then we present the incremental learning and classification procedures of grid- and dummy-cluster-based CCAS methods. The application of two CCA-S methods to intrusion detection is described. The analysis of the testing results is provided.

II. CLUSTERING AND INSTANCE-BASED CLASSIFICATION

A. Clustering

Cluster analysis put data points into clusters based on their similarity. It is mainly used to discover the underlying structure of the data when we do not have any background knowledge about the data. Cluster analysis relies on attribute values of data points to partition data points into clusters such that a given objective function is optimized.

Various methods of solving clustering problems through optimization include greedy search, dynamic programming, divide-and-conquer and so on. Since solving clustering problems through optimization is NP-complete in most cases [7-12], many heuristic algorithms have been developed to produce a good solution that satisfies an attainable bound. Among well-known heuristics algorithms are hierarchical clustering algorithms and partitioning clustering algorithms such as the *K*-means method [7-13]. These algorithms do not necessarily lead to a globally optimal solution. The hierarchical clustering algorithm uses a nested sequence of partitions, which can be agglomerative or divisive. Partitioning algorithms construct a partition of the data points into clusters such that the data points in a cluster are more similar to each other than to the data points in different clusters. Both the hierarchical and partitioning clustering algorithms require a complete matrix of pair-wise distances between all data points before the clustering can proceed. This creates difficulty in incremental learning that must update clusters when new data points become available. To do this, non-incremental clustering algorithms need to re-compute or add new items to the pair-wise distance matrix and start the clustering procedure all over again using the updated matrix. This solution to handling new data points is not efficient when we deal with large amounts of data.

A simple method of incrementally clustering data points is to process data points one by one and group the next data point into an existing cluster that is closest to this data point. However, there exists a problem of "local bias of input order". For example, the first cluster is constructed with only the first data point in a data set. The second data point in the data set is grouped into this existing cluster because it is the closest cluster to this second data point and the only cluster available, even though the second data point may not be similar to the first data point or have a large distance to the first data point in the space. As a consequence, all data points in the data set will end up in a single cluster. Therefore, an incremental clustering method must overcome this problem of "local bias of input order". The following methods to enable scalable, incremental learning have been reported.

Grid-based clustering. In this method the data space is partitioned into a number of non-overlapping regions or grids. A histogram is then constructed to describe the density of the data points in each grid cell. Only data points in the same grid cell can be clustered together. Hence, grid cells in the data space prevent all data points in a data set from being grouped into a single cluster.

Density-based clustering. Density-based clustering uses a local cluster criterion. It requires that a cluster should contain a certain number of points within some radius. Hence, density-based methods consider clusters as regions of data points with high density, and clusters are separated by regions of data with low density or noise.

Subspace clustering. Subspace clustering is a grid-based method. It is a bottom-up method of finding dense units in lower dimensional subspaces and merging them to find dense clusters in higher dimensional subspaces. This method is based on the concept that if a dense cell exists in k dimensions then all its projections in a subset of k dimensions are also dense.

Several algorithms have been developed based on the above methods. BIRCH uses a hierarchical data structure, called CF-tree with thresholds on nodes and leaves, to incrementally cluster data points [7-14]. DBSCAN is a density-based clustering algorithm [7-15]. WaveCluster is a density-based and grid-based technique that applies the wavelet transformation to the feature space of data points [7-10]. MAFIA is a subspace clustering technique that uses the adaptive grid size to improve clustering quality [7-11]. Among these algorithms BIRCH and DBSCAN support the incremental learning ability. All these algorithms have the computation complexity of $O(N)$ with regard to N data points in a data set. Obviously, the computational complexity is also influenced by other specific parameters of these algorithms.

In all the above clustering algorithms, domain knowledge is usually required either during clustering to determine parameters in these algorithms or after clustering to interpret clustering results. For example, using different distance thresholds, a hierarchical clustering tree produces different sets of clusters. This distance threshold is often chosen by examining whether the resulting set of clusters is meaningful according to domain knowledge. With a large number of data points and a large number of resulting clusters, it is often difficult to see whether the set of clusters is meaningful or not.

There are parameters in each of the above clustering methods. For example, the size and number of grid cells are parameters in the grid-based clustering methods. In the density-based method, the radius is a parameter that is critical to clustering results and performance. To choose appropriate values for these parameters, domain knowledge as well as trial-and-error experiments are required.

For clustering problems, only attribute values of data points are available to uncover the clustering structure of data points. However, for classification problems such as intrusion detection, both attribute values and target values of data points in a training data set are available. Target values of data records in the training data set reflect domain knowledge about the data, and can be directly used to supervise the clustering of data points and assign a target value to each cluster. A cluster structure with a target value assigned to each cluster can be used as a classification function to classify new data points. CCAS uses a scalable clustering procedure with additional features to support supervised clustering and incremental learning. The resulting clusters with assigned target values from CCAS represent a classification function.

B. Instance-based Classification

The basis of instance-based classification is that the classification of an instance should be most similar to the classification of other instances that are similar to it [7-16]. For classification we look for *k* nearest neighbors of a data point to be classified. We classify the data point based on classes of its neighbors. *K-nearest-neighbor* methods have been used successfully in many classification problems [7-17]. However, when original data points are used to select *k* nearest neighbors for classification, noise and outliers often have impact on classification performance. It is expected that using clusters of data points helps reduce impact of noise and outliers and thus improve classification performance. Another advantage in using clusters of data points is improvement on computation time to determine k nearest neighbors and thus scalability.

III. CLUSTERING AND CLASSIFICATION ALGORITHM – SUPERVISED (CCAS)

Let each data record be a $(p+1)$-tuple with the predictor variable vector X in p dimensions as $\{X_1, X_2, \ldots, X_p\}$ and target variable Y representing the class of each record. Then a data record I s a point in a p-dimensional space. Each predictor variable is numeric or nominal. Y can be a binary variable with value 0 or 1, or a multi-category nominal variable. Y is known for each record in training data. In classification, Y is determined from predictor variables. In this paper, we describe CCA-S for only the numeric predictor variables and the binary target variable. Variations of CCA-S for dealing with other types of predictor variables and target variable will be presented in future reports.

A. Cluster Representation and Distance Measures

In CCAS algorithms, a cluster L is a summarization of its data points, and is represented by the centroid of all the data points in it, with coordinates XL, the number of data points, N_L, and the class label, YL. XL is calculated as:

$$XL = \frac{\sum_{j=1}^{N_L} X^j}{N_L} \qquad (7-1)$$

where X^j is the coordinates of the jth point in this cluster.

The distance from a data point to a cluster can be calculated using different distance measures. For example, the weighted Canberra distance is:

$$d(X,L)=\sum_{i=1}^{P}\frac{|X_i - XL_i|}{X_i + XL_i}r_{iY}^2 \qquad (7\text{-}2)$$

and the weighted Euclidean distance is:

$$d(X,L)=\sqrt{\sum_{i=1}^{P}(X_i - XL_i)^2 r_{iY}^2} \qquad (7\text{-}3)$$

where X_i and XL_i are the coordinates of the data point X and the cluster L's centroid on the i^{th} dimension, and r_{iY} is the correlation coefficient between the prediction variable X_i and the target variable Y.

B. Training - Supervised Clustering

There are two steps to incrementally group the N data points in the training data set into clusters supervised by the target class information.

Step 1. Scan the training data and compute the parameters.

This step calculates the squared correlation coefficient r_{iY} between each predictor variable X_i and the target variable Y to determine how relevant each predictor variable is for predicting the target class in the target variable. Here we give the incremental formula as follows for n = 1, …, N.

$$r_{iY}^2(N)=\left(\frac{S_{iY}{}^2(N)}{\sqrt{S_{ii}{}^2(N)}\sqrt{S_{YY}{}^2(N)}}\right)^2 \qquad (7\text{-}4)$$

where

$$S_{ii}^2(n)=\frac{n-2}{n-1}S_{ii}^2(n-1)+\frac{1}{n}\left(X_i(n)-\overline{X_i}(n-1)\right)^2$$
$$S_{iY}^2(n)=\frac{n-2}{n-1}S_{iY}^2(n-1)+\frac{1}{n}\left(X_i(n)-\overline{X_i}(n-1)\right)\left(X_Y(n)-\overline{X_Y}(n-1)\right)$$

$$S_{YY}^2(n)=\frac{n-2}{n-1}S_{YY}^2(n-1)+\frac{1}{n}\left(X_Y(n)-\overline{X_Y}(n-1)\right)^2$$
$$\overline{X_Y}(n)=\frac{(n-1)\overline{X_Y}(n-1)+X_Y(n)}{n}$$

$$\overline{X_i}(n)=\frac{(n-1)\overline{X_i}(n-1)+X_i(n)}{n}$$

Step 2. Incrementally group each point in the training data set into clusters.

1. For a new data point *X*, find the nearest cluster *L* to this data point using one distance measure *d(X,L)* given above.
2. If *L* has the same target class as that of *X*, add *X* into *L*, and update the centroid coordinates of *L* and the number of the data points (N_L) in this cluster:

$$XL_i=\frac{N_L XL_i+X_i}{N_L+1}\quad for\ i=1,2,,\dots,p \qquad (7\text{-}5)$$
$$N_L=N_L+1$$

3. Otherwise, create a new cluster with this data point as the centroid; the number of the data points in the new cluster is 1, and the target class of the new cluster is the class of this data point.
4. Repeat 1) to 3) until no data point in the training data set is left.

Hence, CCAS performs a non-hierarchical heuristic procedure based on the distance information as well as the target class information of the data points in the training data set.

C. Two Methods of Splitting the Clusters

The above clustering procedure produces a cluster structure where each cluster contains homogeneous data points, i.e., we can consider the clusters in this cluster structure as normal or intrusive clusters. However, if we only use the procedure shown above, when we look for the nearest cluster for a new data point, there is no limit on the range of the search area. That means this data point can even be incorporated into a cluster far away from it in distance. In this case small clusters by nature tend to combine into large clusters not by nature but due to the input order of training data points. To deal with this problem, we use two different methods to limit the search area and promote the splitting of data points into smaller clusters.

D. Dummy Clusters Used in Initial CCA-S

In our initial CCA-S algorithm, a dummy cluster is created in the first step of training stage for each target class with the centroid coordinates $\{XL_{k1}, XL_{k2}, \ldots, XL_{kp}\}$ being the mean vector of the predictor variables for that target class as follows.

$$XL_{ik} = \frac{\sum_{n=1}^{N_k} X_i(n)}{N_k}, \qquad k = 0,1 \text{ and } i = 1,2,\ldots,p \tag{7-6}$$

where N_k is the number of data points with the target class of k.

The two dummy clusters for the two target classes (0 and 1) respectively are assigned with the same target class (e.g., 2) which is different from the target classes of 0 and 1. As initial clusters, the dummy cluster for a given class (0 or 1) is used in the clustering procedure to let the clusters for this class spread over the entire space of the data population for this class rather than congest around the centroid location of the data population for this class, because data points in the training data set have a different target class (0 or 1) than the target class of these two dummy clusters (2).

The number of produced clusters has impact on time of conducting the clustering procedure and the classification procedure. When we deal with large data sets, we may want to control the number of produced clusters for speeding up the time of clustering and classification. We can use two methods to control the number of clusters produced in the training phase. In one method, we specify a threshold on the number of clusters for each target class. Then during clustering the related dummy cluster is removed after the number of clusters for that target class reaches the given threshold, thus slowing down the increase of clusters for that target class. The other method controls the number of clusters for each class directly. After enough points are used to create cluster representatives for each target class, new data points will only update the existing clusters and no new cluster is created.

E. Grid-based Clustering

The grid-based method is a common method found in existing work to cope with the problem of large clusters not by nature but due to the input order of training data points. This method first divides the space of data points into grid cells. Many methods are available for this purpose. In our study each dimension is divided into a set of equal intervals in the range limited by the minimum and maximum values of data points on this dimension. The number of intervals on each dimension can be different and is decided by users. Then the whole space is separated into "cubic" cells by the grids determined by the end points of these intervals. A cluster belongs to one grid cell and can be assigned a grid index referring to its grid cell.

Based on the grid cells, a supervised incremental clustering can be performed. For each training data point, we search the existing clusters to look for the nearest cluster to this data point in the same grid cell. If the cluster has the same target class as this data point, this data point is grouped into this cluster and the centroid is updated. If there is no cluster in the grid cell of this data point or the target class of the nearest cluster is different, a new cluster is created with this data point as the centroid and the target class is the same as this data point. This process repeats until all data points in training data are incorporated.

There are several methods for searching the clusters in grid cells. In our study, we number the interval of each dimension that bounds a grid cell with an integer. Then for each grid cell we have a set of integers for all dimensions, which construct an index vector for this grid cell. Then we give each cluster or each data point an index vector indicating its grid cell in space. This is the simplest way to use the grids. The disadvantage is that we have to search every cluster in order to find the nearest one to a data point.

F. Incremental Learning

CCAS supports the incremental update of clusters with new training data. Note that formulas (7-4) and (7-5) are given in a recursive form. Hence, the parameters from the previous training phase, including the correlation coefficient for each predictor variable, the centroid coordinates of each cluster, and the number of the data points in each cluster, are kept all the time and thus can be updated incrementally with each of the new training data points by repeating the steps in training.

We see that grid is not impacting the incremental learning ability of the clustering procedure. We only need to record the value range and the number of grid intervals in each dimension. If we use dummy clusters, there is a recursive form for calculating the centroid coordinates of dummy clusters in training. For $c,j \in \{0,1\}$, $j \neq c$, $i=1,\ldots,p$, and N_j and N_c are the numbers of data points in dummy cluster j and c respectively, if current data point belongs to class c, we calculate equation (7-7):

$$\begin{cases} XL_{ic}(n) = \dfrac{N_c(n-1)XL_{ic}(n-1) + X_i(n)}{N_c(n-1)+1} \text{ and } N_c(n) = N_c(n-1)+1 \\ XL_{ij}(n) = XL_{ij}(n-1) \text{ and } N_j(n) = N_j(n-1) \end{cases} \tag{7-7}$$

From this equation, we can realize the incremental learning ability for the splitting method based on dummy clusters. In fact, once we have an initial cluster structure, we do not need to store the dummy clusters.

G. Testing - Classification

The above supervised clustering, with the method of either dummy clusters or grids, maps the space of training data points into a cluster structure where normal and intrusive clusters occupy different locations. These clusters represent signature patterns of normal or intrusive activities in a computer system. Based on instance-based classification, we classify a new data point by comparing the data point with the clusters of signature pattern. We assign the distance-weighted average of the target values of the k nearest clusters, L_1, ..., L_k, to the target value of the data point X as follows:

$$W^j = \frac{1}{d^2(X, L^j)}$$

$$Y = \frac{\sum_{j=1}^{k} YL^j W^j}{\sum_{j=1}^{k} W^j} \tag{7-8}$$

where L_j is the jth nearest cluster, W^j is the weight for the cluster L^j based on the distance from X to the centroid of this cluster; the target class of this cluster is YL^j, and the target class of the X data point is Y. The class value Y of this data point falls in the range of [0,1] to describe the closeness of this data point to the two target classes of 0 and 1 as presented in the k nearest clusters of this data point.

Parameter k is chosen by users. In our study when we use CCAS based on dummy clusters, we use all the clusters produced from the training stage to calculate the target value of a new data point during the testing stage. It is not appropriate to do this when we use the clusters produced from the grid-based clustering, where the grid structure limits the growth of clusters. However, in each grid cell the clustering operation is carried out in a normal manner. Our experiment result does not show good performance when all the clusters from the grid-based method are used for classification of a new data point during the testing stage. Therefore, when we use the grid-based CCAS to produce clusters from the training stage, we use only the clusters in the same grid cell of a new data point to calculate its target value. This is a strategy to define the classification neighborhood as a region with fixed size around the new data point. If there is no cluster in this grid cell, we will search for the nearest cluster in the whole cluster structure to the new data point and assign the class of this cluster to the target value of this point.

H. Computation Complexity

Let the number of points in training data be N. The number of predictor variables is p. The number of output clusters after each phase is M. The first step in training is to scan the data points in the training data, which has the computation complexity $O(pN)$. For the incremental clustering in the second step, the upper bound on the computation cost is $O(pNM)$ if we search the produced cluster structure sequentially. Similar to other scalable clustering algorithms such as BIRCH and DBSCAN, the overall complexity is $O(N)$ with respect to the number of training data points. In fact we can apply efficient storage management of the produced clusters. For example, we can store clusters based on grid cells. If we use B intervals in each dimension and assume that the produced clusters are evenly distributed among grid cells, the average number of clusters in one grid cell is about $\frac{M}{B^p}$. When clustering each data point, we only search the clusters in the grid cell of this data point. The computation cost in this case is $O(pN\frac{M}{B^p})$. Of course the number of grid cells containing clusters is usually much less than the total number of grid cells, and then computation cost will be greater than this estimate. Using storage management techniques such as that in [18], this computation cost is improved to almost linear to $O(pN)$. The computation cost of classifying a data point is $O(pM)$. Again this computation cost can be reduced to almost $O(p)$ if we use efficient technique to store and search the cluster structure.

IV. APPLICATION TO INTRUSION DETECTION

CCAS is tested on a large data set of computer activity data for computer intrusion detection. We want to investigate possible effects of different input orders of training data points when two methods of CCAS are used to split clusters in supervised clustering. We will examine prediction accuracy and robustness of the two different CCAS methods. In this application of CCAS to computer intrusion detection, we use weighted Euclidean distance in clustering and classification. In the grid-based method, we set the number of grid intervals in each dimension as 11. We do not control the growth of clusters when using dummy clusters.

A. Data Representation

Computer audit data used in our study consist of audit events sequenced in time order from a computer host machine. These audit data are recorded by the Basic Security Module (BSM) in a Solaris operating system. Each audit event has an audit record. Each audit record has a number of attributes, including event type, user ID, process ID, command type, time, remote IP address, and so on. We add the target class to each audit record in the training data set, which is 1 if the corresponding audit event is a part of an attack and 0 otherwise. We also know the ground truth (normal or intrusive) of each audit record in the testing data set, but let CCAS predict the target value of each audit record in the testing data set. Then the predicted target value of each audit record in the testing data set is compared with the true target class of that audit record for evaluating prediction accuracy of CCAS.

Event type is an important characteristic of activities in a computer system, and has been successfully used for computer intrusion detection in many existing studies. In our study we also use this information of event type for intrusion detection. That is, we extract only event type information from each audit record of an audit event. There are totally 284 different types of auditable event in this Solaris system. We use 284 predictor variables to represent the frequencies of the 284 event types respectively that occur in the recent past of the current audit event. Given a stream of audit events, we use an exponentially weighted moving average (EWMA) technique to obtain a smoothed frequency distribution of event types in the recent past of the current audit event. Considering the t^{th} event as the current event, we use the following formula to compute the smoothed frequency distribution of the 284 event types in the recent past of the t^{th} event:

$$\begin{cases} X_i(t) = \lambda \times 1 + (1-\lambda) \times X_i(t-1) \textit{ if the current event is the ith event type} \\ X_i(t) = \lambda \times 0 + (1-\lambda) \times X_i(t-1) \textit{ if the current event is not the ith event type} \end{cases} \tag{7-9}$$

where $X_i(t)$ is the smoothed observation value of the i^{th} predictor variable for the current event, and λ is a smoothing constant which determines the decay rate.

The value of each predictor variable not only captures the information of the current event, but also reflects the information of the recent events. With this EWMA method, we add the time characteristic into the values of the predictor variables to represent the frequency distribution of event types in the recent past of the current event. In our study, we initialize $X_i(0)$ to be 0 for i = 1,...,284. We let λ be 0.3—a common value for the smoothing constant [19]. Hence, for each event in the training and testing data set, we obtain a vector of $(X_1, \ldots, X_{284})$.

B. Data Sets

We use the MIT Lincoln Laboratory's 2000 DARPA Intrusion Detection Evaluation Data (http://ideval.ll.mit.edu) to create our training and testing data. The data were generated in a local-area network (LAN) at the MIT Lincoln Laboratory by simulating activities in a typical U.S. Air Force LAN. The data include several phases of a distributed denial of service (DDoS) attack. This attack scenario is carried out over multiple network sessions. These sessions have been grouped into 5 attack phases, over the course of which the attacker probes, breaks-in via some Solaris vulnerability, installs trojan DDoS software, and finally launches a DDoS attack. There are 15 normal sessions and 7 attack sessions in the data stream from the host machine called "Mill," and 63 normal and 4 attack sessions in the other data stream from host machine "Pascal". Each of the two event streams in the 2000 data has over a hundred thousand audit records. We use the data from "Pascal" as the training data and the data from "Mill" as testing data.

We want to observe possible performance differences between the two methods of CCAS with respect to robustness to the input order of training data points. In the 2000 DARPA Intrusion Detection Evaluation Data, audit events occur in different sessions and these sessions are recorded in almost sequential manner. In the original input order of the data, the attack sessions are mixed in the middle of normal sessions. In addition to the original input order, we create other three different input orders of the same data by rearranging sessions while maintaining the order of audit records in each session. Table 7-1 shows the four input orders.

Table 7-1. Four input orders of training data.

Input order	**Session orders**
1	Original order of sessions
2	Reversed order of sessions
3	Normal sessions followed by attack sessions
4	Attack sessions followed by normal sessions

V. ANALYSIS OF RESULTS

Two types of errors occur when using an intrusion detection algorithm. A false positive occurs when the algorithm predicts an event as intrusive when it is in fact normal. A false negative (or a false alarm) occurs when a truly intrusive event occurs but the algorithm allows it to pass as a normal event.

In this study the target value of each audit record in the testing data falls in [0, 1] and indicates the intrusiveness of the event. Given a signal threshold, we signal an event as intrusive if the target value is greater than or equal to this signal threshold; otherwise we claim the event as normal. A signal on a truly normal event is a false alarm. A signal on a truly intrusive event is a hit. A false alarm rate is the ratio of the number of signals on all the normal events in the testing data to the total number of all the normal events in the testing data. A hit rate is the ratio of the number of signals on all the intrusive events in the testing data to the total number of all the intrusive events in the testing data. Hence, given a signal threshold, we obtain a pair of the false alarm rate and the hit rate. We then plot a ROC (Receiver Operating Characteristics) curve using pairs of false alarm rate and hit rate that are generated by varying the value of the signal threshold. In a ROC curve, *X*-axis represents the false alarm rate while *Y*-axis represents the hit rate. The closer the curve is to the top-left corner (100% hit rate and 0% false alarm rate) of the plot, the better the detection performance. Such a ROC analysis is based on the target value of each audit event in the testing data for evaluating detection performance on individual audit events. Hence, it is called the event-based ROC.

In this study we evaluate detection performance on sessions and perform session-based ROC analysis by obtaining a session signal ratio for each session in the testing data. The session-based ROC analysis includes the following steps:

1. Calculate a signal threshold to determine if we should signal an individual event based on its predicted target value. There are several ways to determine this signal threshold. In our study we calculate the average μ and the standard deviation σ of predicted target values for all normal the events in the training data. The signal threshold is set as $\mu + a\sigma$, because we are mainly interested in detecting a large predicted target value close to 1 which indicates a possible intrusion, where a is a coefficient to adjust the signal threshold.
2. Compute a "session table" containing a session signal ratio for each session. For each session we calculate the ratio of the number of signals to the total number of events in the session as the session signal ratio for a given signal threshold.
3. Plot the ROC curve based on the session signal ratios of all the sessions in the testing data.

We expect that session signal ratios for normal sessions are lower than session signal ratios for intrusive sessions. The choice of a influences detection performance. If we consider that normal activities produce data points in a certain region of the data space, then $a\sigma$ is a measurement of the deviation from this "normal" region. We see that a is specific to the data set if we want to choose the "best" signal threshold for determining whether a data point is normal or not. In this study, we try various values of a, and it appears that small a values of 0.5, 0.6, and 0.7 produce the best performance.

Figure 7-1 shows the ROC curves on the data in input order 1 for grid- and dummy-cluster-based CCAS methods. It can be seen that CCAS based on dummy clusters detects at most 71.4% of attack sessions at the level of 0% false alarm rate. If we lower the requirement for the false alarm rate, it even detects all the attack sessions with the false alarm rate less than 10%

using the *a* value of 0.5. The performance of the grid-based CCA-S method is worse. At the level of 0% false alarm rate, it detects 57.1% of attack sessions at most.

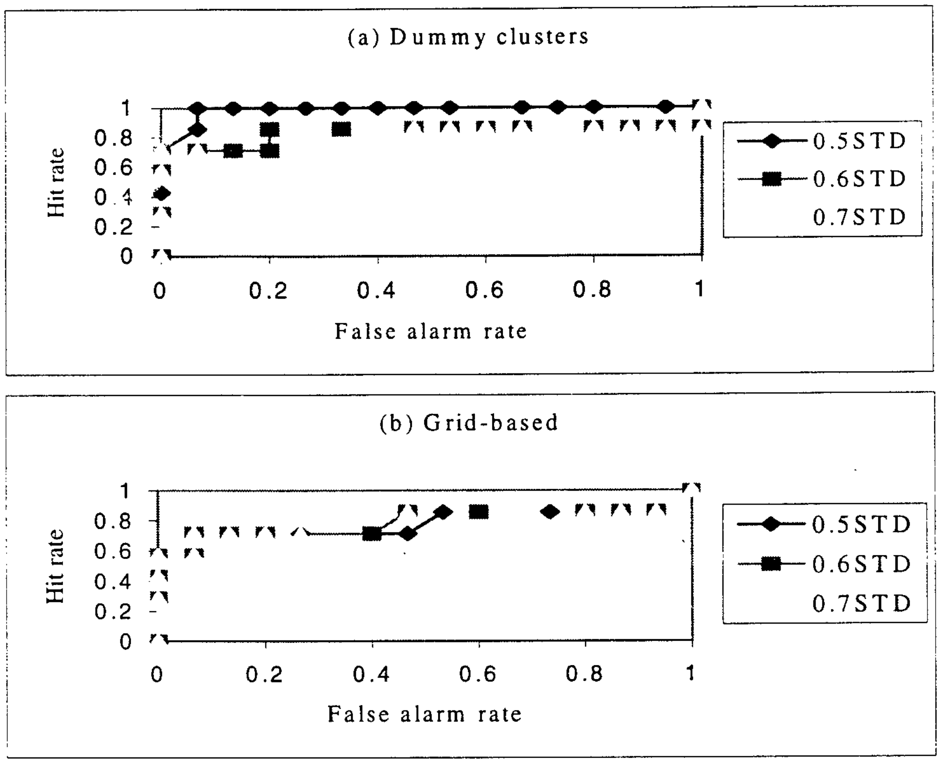


Figure 7-1. ROC curves for input order 1.

Figure 7-2 shows the ROC curves for input order 2. Both CCAS methods appear to have similar performance. The performance of CCA-S based on dummy clusters is slightly better than that of the grid-based CCAS. Performance of both methods seems slightly worse than their performance for input order 1.

As shown in Figure 7-3, the performance of both methods drops for input order 3. The performance of CCA-S based on dummy clusters drops significantly even below that of grid-based CCAS. The performance of grid-based CCA-S is just slightly worse than its performance for input orders 1 or 2. At the false alarm rate of 6.67%, grid-based CCAS still correctly detect 57.1% of attack sessions.

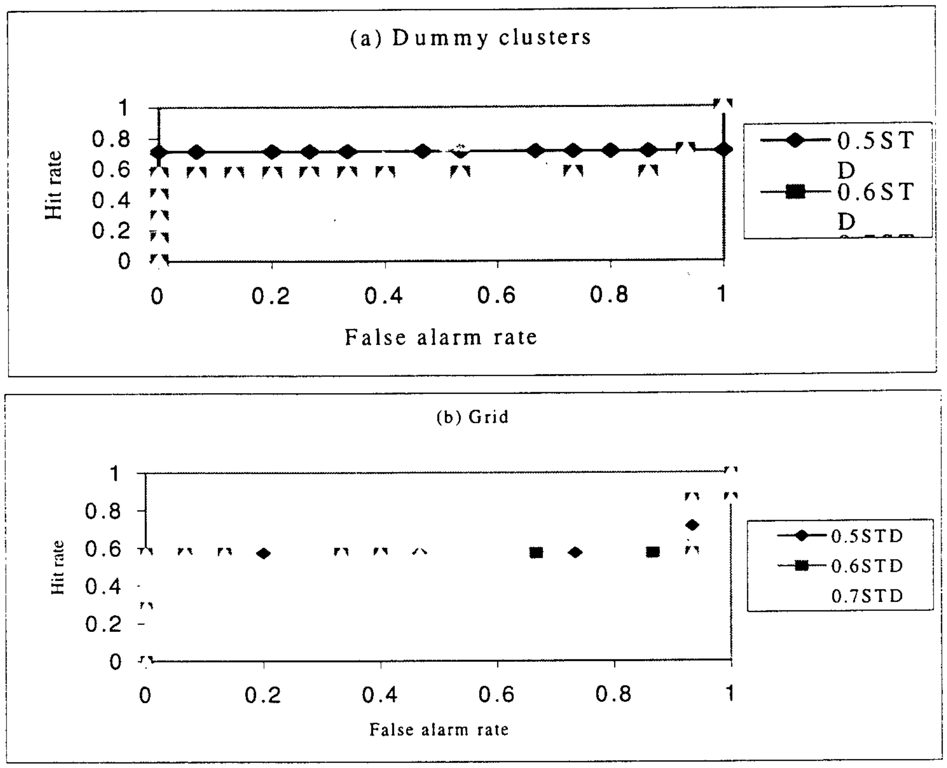


Figure 7-2. ROC curves for input order 2.

A possible explanation for the worse performance of CCAS based on dummy clusters is provided below. If we assume that normal data points have a multivariate normal distribution, whereas intrusive data points scatter in the multi-dimensional space as outliers to this multivariate normal distribution lying at the outside edge of the normal region. With input order 3 of the training data, all the normal data points are considered for clustering first. If we use dummy clusters, those data points at the outer range of the normal region tend to be merged into the clusters near the centroid of the normal data distribution and produce much fewer normal clusters than those produced for other input orders, as we have observed from the statistics of the number of normal and intrusive clusters produced from training. For the normal data points in the testing data that fall at the outer range of the normal region, there may not be any normal clusters in their neighborhood. Therefore, those normal data points might have been considered closer to intrusive clusters and have been classified as intrusive, thereby producing more false alarms. This may explain why the false alarm rate of CCAS based on dummy clusters is high while we try to improve the hit rate. The grid-based CCAS method uses grids to overcome this problem with this input order by forcing the normal clusters to split between grid boundaries.

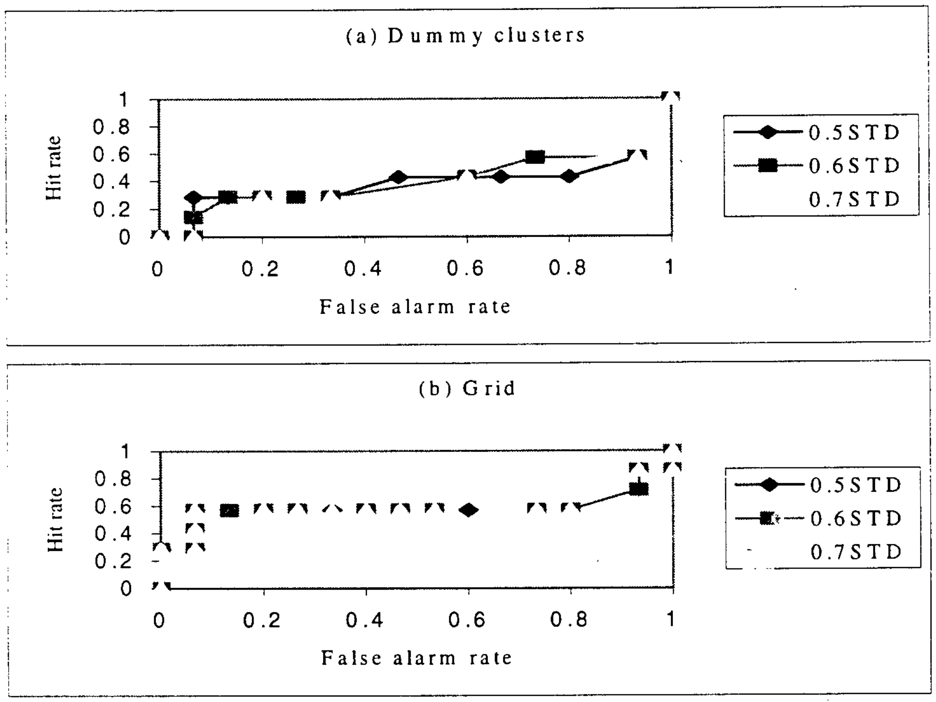


Figure 7-3. ROC curves for input order 3.

Figure 7-4 shows the performance of both methods for input order 4. CCAS based on dummy clusters detects at most 85.7% of attack sessions at the level of 0% false alarm rate. This indicates that 6 of 7 attack sessions in the testing data are detected with no false alarms. The grid-based CCA-S method detects at most 71.4% of attack sessions at the level of 0% false alarm rate.

That indicates that 5 of 7 attack sessions are detected. If we lower the requirement for the false alarm rate, it detects all attack sessions with the false alarm rate of 6.67%.

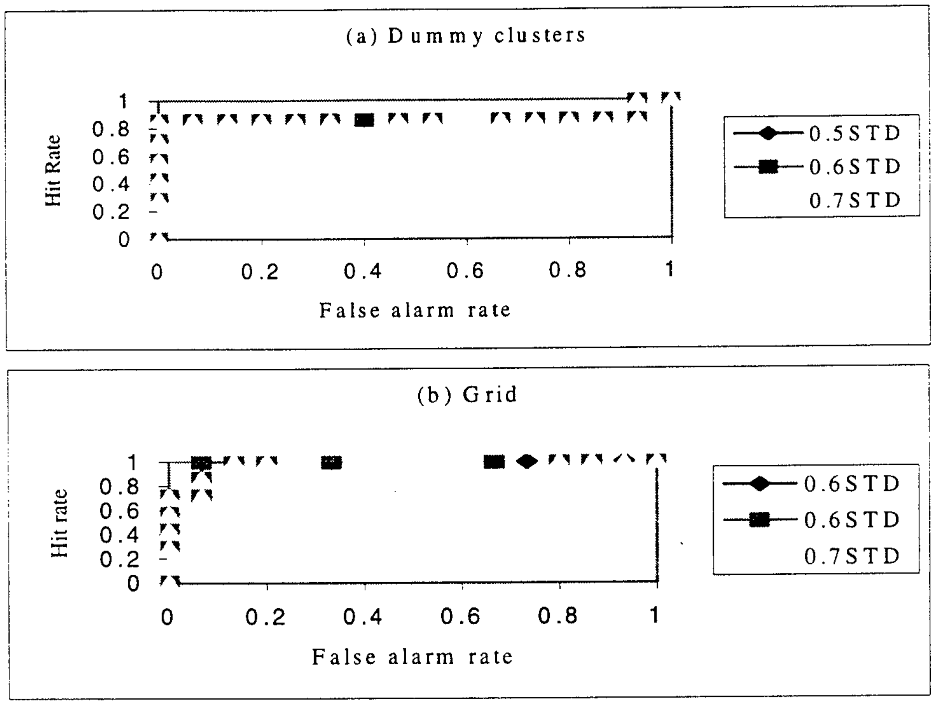


Figure 7-4. ROC curves for input order 4.

In overall, the number of clusters produced by the grid-based CCAS method is much greater than the number of clusters produced by CCAS based on dummy clusters. The number of clusters for the latter is in hundreds while the former produces thousands of clusters. Obviously the number of clusters is also impacted by the parameter—the number of grid intervals.

When comparing the number of clusters produced from different input orders of the training data, it appears that input order 3 leads to the smallest number of normal clusters, whereas the smallest number of intrusive clusters is produced from input order 4 in which all intrusive data points are considered first for clustering. If we consider effects of different methods of splitting clusters, the change in the number of clusters for the grid-based CCAS method from one input order to another is very small whereas the change in the number of clusters for CCA-S based on dummy clusters is large.

VI. CONLUSIONS

In this chapter we present advantages and disadvantages of two different methods (grid-based and dummy-cluster-based) of splitting clusters while performing CCAS procedures for clustering and classification. The testing results show that different input orders of training data have certain impact on performance of both methods. The impact on the performance of CCAS based on dummy clusters is more significant when all normal data are presented first before attack data in the training data set. Grid–based CCAS produce more clusters than CCAS based on dummy clusters, implying that grid-based CCAS takes more computation time in clustering and classification. CCAS based on dummy clusters produce better performance than grid-based CCAS for three of the four input orders, and in overall produce fewer clusters. Therefore, we recommend CCAS based on dummy clusters with respect to detection performance and computation cost. However, special caution must be taken to avoid the arrangement of having all normal data before all attack data in the training data set. A training data set with a mixture of normal data and attack data in the input order is more desirable.

REFERENCES

[7-1] Debar H, Dacier M, Wespi A. Towards a Taxonomy of Intrusion-detection Systems. *Computer Networks* 1999 **31**: 805-822.

[7-2] Axelsson S. *Intrusion Detection Systems: A Survey and Taxonomy*; Department of Computer Engineering, Chalmers University of Technology, Goteborg, Sweden, 2000.

[7-3] Ye N, Li X, Emra SM. Decision Tree for Signature Recognition and State Classification. *Information Assurance and Security Workshop of IEEE Systems, Man, and Cybernetics*; West Point, New York, USA, 2000.

[7-4] Lee W, Stolfo SJ, Mok K. A Data Mining Framework for Building Intrusion Detection Models. *Proceedings of 1999 IEEE Symposium on Security and Privac* 1999; pp 120-132.

[7-5] Sinclair C, Pierce L, Matzner S. An Application of Machine Learning to Network Intrusion Detection. *Proceedings of 15th Annual Computer Security Applications Conference (ACSAC '99)* 1999; pp 371-377.

[7-6] Endler D. Intrusion Ddetection: Applying Machine Learning to Solaris Audit Data. *Proceedings of 14th Annual Computer Security Applications Conference* 1998; pp 268–279.

[7-7] Valdes A. Skinner K. Adaptive, Model-based Monitoring for Cyber Attack Detection. *Recent Advances in Intrusion Detection (RAID) 2000*; Toulouse, France.

[7-8] Ye N and Li X. A Scalable Clustering Technique for Intrusion Signature Recognition. *The 2nd Annual IEEE Systems, Man, and Cybernetics Information Assurance Workshop*; West Point, New York, 2001.

[7-9] Ye N, Li X. Scalable Clustering Techniques for Quality Assurance of Information Systems. *INFORMS San Antonio 2000;* Session MD15; San Antonio, Texas, USA, 2000.

[7-10] Sheikholeslami G, Chatterjee S, Zhang A. WaveCluster: A Multi-Resolution Clustering Approach for Very Large Spatial Databases. *Proceedings of 24th VLDB Conference* 1998; New York, USA.

[7-11] Harsha SG, Choudhary A. *Parallel Subspace Clustering for Very Large Data Sets*; Techinical report: CPDC-TR-9906-010; Northwestern University, 2000.

[7-12] Procopiuc CM. *Clustering Problems and Their Applications (a survey)*; Department of Computer Science, Duke University, 1997.

[7-13] Jain AK, Dubes RC. *Algorithms for Clustering Data*; Prentice Hall, 1988.

[7-14] Zhang T. *Data Clustering For Very Large Datasets Plus Applications*; Ph.D. Thesis; Department of Computer Science, University of Wisconsin – Madison, 1997.

[7-15] Ester M, Kriegel HP, Sander J, Wimmer M, Xu X. Incremental Clustering for Mining in a data Warehousing Environment. *Proceedings of 24th VLDB Conference,* 1998; New York, USA.

[7-16] Mitchell T. M*achine Learning*; WCB/McGraw-Hill, 1997.

[7-17] Cherkassky V, Mulier F. *Learning from Data*; John Wiley & Sons, 1998.

[7-18] Huang C, Bi Q, Stiles R, Harris R. Fast Search Equivalent Encoding Algorithms for Image Compression Using Vector Quantization. *IEEE Trans. on Image Processing* **1**(3): 413-416.

[7-19] Ryan TP. *Statistical Methods for Quality Improvement*; New York: John Wiley & Sons, 1988.

Chapter 8 Post-Processing of Cluster Structure for Robust Application of CCAS to Intrusion Detection

In Chapter 7, we present how CCAS is used to learn signature patterns of normal and intrusive activities from training data and use these signatures to classify observed system activities as normal or intrusive. This chapter presents an extension of CCAS, called robust CCAS, which aims at improving robustness of CCAS to input order of training data and noises and thus performance of CCAS for detecting intrusions into computer and network systems. The robust CCAS adds the redistribution of training data points, the supervised hierarchical clustering of clusters, and the removal of outliers as the post-processing steps after steps in the CCAS produce clusters of training data points. The robust CCAS is tested using a large set of computer audit data for intrusion detection. The testing results show that the robust CCAS makes significant improvement in detection performance and robustness to input order of training data and noises.

I. INTRODUCTION

Intrusions into computer and network systems have presented significant threats to these systems for providing continued service. Many intrusions into computer and network systems are attributed to vulnerabilities and bugs in system and application software. While work on software engineering continues to improve quality of software and minimize vulnerabilities and bugs, intrusion detection has also become an important part of assuring quality of software and its provided service through on-line monitoring of system activities and detection of any attempts to exploit software vulnerabilities and bugs to break into computer and network systems.

There exist two general categories of intrusion detection techniques: anomaly detection and signature recognition (misuse detection) [8-1 – 8-9]. Anomaly detection techniques first build a profile of a subject's normal activities and then signal observed activities as intrusive if they deviate significantly from the profile of normal activities. Many signature recognition techniques first learn signature patterns of normal and intrusive activities from training data, and then match these signature patterns with incoming data of observed activities to determine whether the activities are normal or intrusive. Anomaly detection is capable of detecting novel attacks, while signature recognition is accurate in detecting known attacks. Anomaly detection and signature recognition often co-exist in an intrusion detection system to complement each other.

This chapter focuses on signature recognition techniques for intrusion detection. Intrusion detection through signature recognition can be considered as a classification problem. A set of attribute or predictor variables, $X = (X_1, X_2, \ldots, X_p)$, can be used to describe normal and intrusive activities. A target variable, Y, can be used to represent class (normal or intrusive) of activities. Given a data record of observed activities, intrusion detection is to assign a value to the target variable of this data record indicating whether this data record is considered as normal or intrusive. Intrusion detection relies on a classification function, $f: X \rightarrow Y$, which can be learned from training data.

Many data mining techniques, such as decision trees [8-10, 8-11], association rules [8-12], artificial neural networks and genetic algorithms [8-13, 8-14], and Bayesian networks [8-15], have been used for intrusion detection through signature recognition. Those data mining techniques are used to learn signature patterns of normal and intrusive activities from training data, and use those signature patterns to classify data of observed activities as normal or intrusive.

However, these existing data mining techniques are not capable of learning signature patterns from large amounts of activity data from computer and network systems in a scalable, incremental manner. First of all, activity data from a computer and network system can easily contain millions of records per day. In addition, each record may have hundreds of data fields. A data mining algorithm to learn signature patterns from such large amounts of data must be scalable. Secondly, patterns of normal and intrusive activities likely change over time, and new forms of attacks emerge constantly. Hence, a data mining algorithm must have the incremental learning ability to update signature patterns as more training data of normal and intrusive activities become available.

We have developed an innovative data mining algorithm [8-16 – 8-19], called Clustering and Classification Algorithm – Supervised (CCAS), which has the scalable, incremental learning ability. CCAS is based on two concepts: supervised clustering for learning signature patterns of normal and intrusive activities during a training phase, and instance-based classification for using signature patterns of normal and intrusive activities to classify observed activities during a testing phase. During the training phase, the value of the target variable for each data record in the training data is known and is used to supervise the clustering of normal and intrusive data records. The resulting clusters represent signature patterns of normal and intrusive activities in the training data, and are used during the testing phase for classifying data of observed activities.

However, just like many other data mining algorithms with the incremental learning ability, CCAS shows some sensitivity to input order of training data because CCAS updates the cluster structure incrementally while processing training data points in a sequential order [8-19]. To overcome this problem, we develop an extension of CCAS, called robust CCAS, which aims at improving robustness of CCAS to input order of training data and noises and thus performance of CCAS for detecting intrusions into computer and network systems. The robust CCAS adds the redistribution of training data points, the supervised hierarchical clustering of clusters, and the removal of outliers as the post-processing steps after steps in the CCAS produce clusters of training data points.

This chapter presents the robust CCAS, its application to intrusion detection, and its performance in intrusion detection. In section II, CCAS is review to provide the basis of describing the post-processing steps in the robust CCAS. Section III presents the algorithm of the robust CCAS. Section IV describes the application of the robust CCAS to intrusion detection and computer audit data used to test detection performance of the robust CCAS. Section V presents and discusses the testing results. Section VI concludes the paper.

II. REVIEW OF CCAS

CCAS is built on a number of concepts in traditional and scalable cluster analysis [8-20 – 8-26] and instance-based classification [8-27 – 8-28] together with several innovative concepts that we develop. A detailed comparison of CCAS with existing algorithms of cluster analysis and instance-based classification and a detailed discussion of the scalable, incremental learning ability of CCAS are provided in [8-19]. In this section, we briefly review steps of CCAS so that in the next section we can present an extension of CCAS that leads to the robust CCAS.

Let each data record be a $(p+1)$-tuple with the predictor variable vector X in p dimensions as $\{X_1, X_2, \ldots, X_p\}$ and the target variable Y representing the target class of each record. Then a data record is a data point in a p-dimensional space. Each predictor variable is numeric or nominal. Y can be a binary variable with value 0 or 1, or a multi-category nominal variable. Y is known for each record in training data. In classification, Y is determined from the predictor variables. In this paper, we describe CCA-S for only numeric predictor variables and binary target variable. Variations of CCA-S for dealing with other types of predictor variables and target variable will be presented in future reports.

A. Cluster Representation and Distance Measures

In CCAS, a cluster L containing a number of data points is represented by the centroid of all the data points in it, with coordinates XL, the number of data points, N_L, and the target class, YL. XL is calculated as:

$$XL = \frac{\sum_{j=1}^{N_L} X^j}{N_L} \qquad (8\text{-}1)$$

where X^j is the coordinates of the jth point in this cluster.

The distance from a data point to a cluster can be calculated using different distance measures. The weighted Euclidean distance is used in this study of applying CCAS to intrusion detection:

$$d(X,L) = \sqrt{\sum_{i=1}^{P} (X_i - XL_i)^2 r_{iY}^2} \qquad (8\text{-}2)$$

where X_i and XL_i are the coordinates of the data point X and the cluster L's centroid on the ith dimension, and r_{iY} is the correlation coefficient between the prediction variable X_i and the target variable Y.

In the extension of CCAS discussed in the next section, we also use the distance between two clusters that is calculated using the above distance measure except that the coordinates of a data point are replaced by the coordinates of a cluster centroid. For cluster *j* and *k*, the weighted Euclidean distance is:

$$d(X,L)=\sqrt{\sum_{i=1}^{P}(XL_i^j - XL_i^k)^2 r_{iY}^2} \tag{8-3}$$

where XL_i^j and XL_i^k are the *i*th coordinates of the centroids of clusters *j* and *k* respectively.

B. Training - Supervised Clustering

There are two steps to incrementally group *N* data points in the training data set into clusters, which is supervised by the target variable information. CCA-S performs a non-hierarchical clustering procedure based on the distance information as well as the target class information of the data points in the training data set.

Step 1. Scan the training data and compute necessary training parameters.

This step calculates the squared correlation coefficient r_{iY} between each predictor variable X_i and the target variable *Y* to determine how relevant each predictor variable is for classifying the target class in the target variable. For training data points $n = 1, \ldots, N$,

$$r_{iY}^2(n)=\left(\frac{S_{iY}(n)}{\sqrt{S_{ii}(n)}\sqrt{S_{YY}(n)}}\right)^2 \tag{8-4}$$

where variance $S_{ii}(n)$ and $S_{yy}(n)$, and covariance $Siy(n)$ can be calculated incrementally as shown in [8-19].

Step 2. Incrementally group each data point in the training data set into clusters

1. For a data point *X*, find the nearest cluster *L* to this data point using the distance measure *d(X,L)* given above.
2. If *L* has the same target class as that of *X*, add *X* into *L*, and update the centroid coordinates of *L* and the number of the data points (N_L) in this cluster:

$$XL_i = \frac{N_L XL_i + X_i}{N_L + 1} \quad for\ i = 1, 2, ,..., p \qquad \text{(8-5)}$$
$$N_L = N_L + 1$$

3. Otherwise, create a new cluster with this data point as the centroid; the number of the data points in the new cluster is 1, and the target class of the new cluster is the class of this data point.
4. Repeat 1) to 3) until no data point is left in the training data set.

C. Method of Splitting the Clusters

The above clustering procedure produces a cluster structure where each cluster contains homogeneous data points, i.e., we can consider the clusters in this cluster structure as normal or intrusive clusters. However, if we only use the above procedure, when we look for the nearest cluster for a new data point there is no limit in the range of the search area. It means that this data point can even be grouped into a cluster far away from it in distance. In this case small clusters by nature tend to combine into large clusters not by nature but due to the input order of training data points. To deal with this problem, CCAS may use one of two different methods to limit the search area and enforce the splitting of data points into smaller clusters: the dummy clusters method and the grid-based method [8-19].

In this study of applying CCAS to intrusion detection, we use the grid-based method. This method first divides the space of data points into grid cells. Many methods are available for this purpose. In this study each dimension is divided into a set of equal intervals in the range limited by the minimum and maximum values of data points on this dimension. The number of intervals on each dimension can be different and is decided by users. Therefore, the whole data space is divided into "cubic" cells by the grids determined by the end points of these intervals. A cluster belongs to one grid cell and can be assigned a grid index referring to its grid cell.

Based on the grid cells, a supervised incremental clustering can be performed. For each training data point, we search the existing clusters to look for the nearest cluster to this data point in the same grid cell. If the cluster has the same target class as this data point, this data point is grouped into this cluster and the centroid is updated. If there is no cluster in the grid cell of this data point or the target class of the nearest cluster is different, a new cluster is created with this data point as the centroid and the target class is the same as this data point. This process repeats until all data points in the training data set are incorporated.

D. Testing - Classification

The above supervised clustering procedure maps the training data points into a cluster structure with clusters of different classes. For example, for intrusion detection a cluster structure from the training stage contains normal and intrusive clusters that represent signature patterns of normal or intrusive activities in a computer and network system. Based on instance-based classification, we classify a new data point by comparing the data point with the clusters. We assign the distance-weighted average of the target values of the k nearest clusters, $L_1, \ldots, L_k$, to the target value of the data point X as follows:

$$W^j = \frac{1}{d^2(X, L^j)}$$

$$Y = \frac{\sum_{j=1}^{k} YL^j W^j}{\sum_{j=1}^{k} W^j} \tag{8-6}$$

where L_j is the jth nearest cluster, W^j is the weight for the cluster L^j based on the distance from X to the centroid of this cluster; the target class of this cluster is YL^j, and the target class of the X data point is Y. The class value Y of this data point falls in the range of [0,1] to describe the closeness of this data point to the two target classes of 0 and 1.

III. ROBUST CCAS

Our study reported in [8-19] indicates that CCA-S shows some sensitivity to input order of training data points. One cause of the robustness problem is that the clusters are incrementally formed by incorporating training data points one by one. A data point is incorporated into an existing cluster structure that considers only the training data points processed so far. That is, the existing cluster structure reflects only a local view based on the training data points that have been processed but not all the training data points. Presenting the training data points in a different order may result in a quite different cluster structure, and consequently different performance in classifying new data. Although dividing the data space into grid cells and allowing the formation of clusters only within grid cells help alleviate the robustness problem with input order of training data points, classification performance still shows some sensitivity to input order of training data points as shown in [8-19].

Therefore, we extend CCAS with additional steps to address the robustness problem, including redistributing training data points to refine the clusters, hierarchically clustering the produced clusters, and removing outliers. These additional steps apply to the produced clusters from basic steps in CCAS. Hence, these steps are called post-processing steps. Table 8-1 summarizes problems that are addressed by these additional steps in the robust CCAS and the grid-based clustering method in CCAS.

Table 8-1. Problems addressed by the additional steps in the robust CCAS and the grid-based clustering method in CCAS.

	Incremental learning	Scalability	Robustness
Grid-based clustering	✓	✓	✓
Redistribution			✓
Hierarchical clustering		✓	✓
Outlier removal			✓

A. Redistribution of Training Data Points

Using the redistribution method, all data points in the training data set are clustered again using the clusters produced from CCAS as the initial clusters which are called the seed clusters. With the seed clusters, we start the training steps of CCAS all over again to process all the training data points in a sequential manner. The clustering procedure is same as that in CCAS, except that when a seed cluster is found to be the nearest cluster to a data point, the seed cluster is discarded and is replaced with a new cluster with the data point as the centroid of the new cluster. In the redistribution method, we allow new clusters to emerge and thus allow adjustments of the cluster structure. Obviously this redistribution process can be repeated for many times. Classification performance is expected to improve with repetition but at computation cost. Our experiments show that usually one time of redistribution is sufficient for improving robustness.

B. Hierarchical Clustering of Produced Clusters

In the grid-based clustering method of CCAS, the search for the nearest cluster to a data point considers only the data point's grid cell. However, a natural cluster may correspond to several produced clusters falling into different grid cells respectively in the neighborhood. Hence, we employ a supervised hierarchical clustering procedure to regroup the produced clusters. This supervised hierarchical clustering algorithm is different from the traditional hierarchical clustering algorithms [6-20 – 6-21] in that the supervised hierarchical clustering algorithm combines only a pair of clusters which not only are closest to each other but also have the same target class into one larger cluster. Figure 8-1 illustrates this concept in one dimensional space. Each point represents the centroid of a cluster. They are in two classes indicated by different shapes. The points (original clusters) in each circle have the same class and are closest to each other. They are grouped into one larger cluster. In Figure 8-1 the closest points with different target classes cannot be combined.

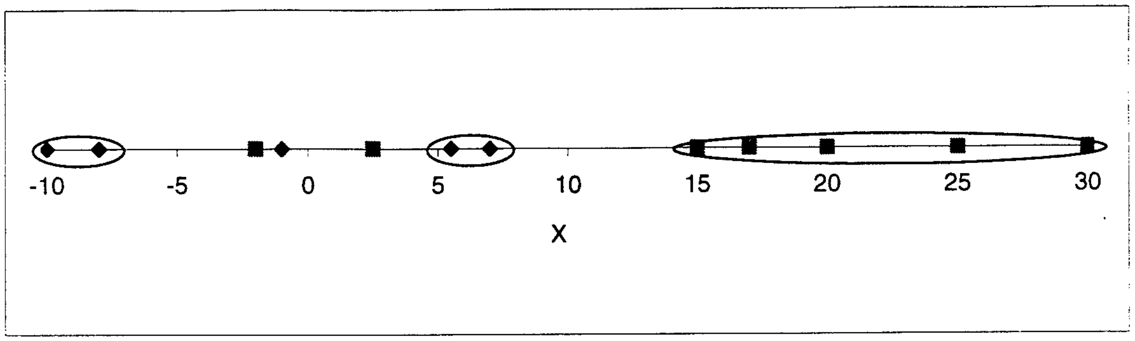


Figure 8-1. An illustration of the supervised hierarchical clustering algorithm.

In our supervised hierarchical clustering algorithm, a single linkage method [8-20 – 8-21] is used in calculating the distance between two clusters, where the distance between two clusters is defined as the distance between their nearest "points". These points are the produced clusters from CCAS. The distance of two produced clusters is the distance between their centroids.

The application of the supervised hierarchical clustering algorithm to the produced clusters from CCAS not only improves the robustness of the cluster structure but also reduces the number of the resulting clusters. This results in improvement in computation time to perform classification and thus improvement in scalability.

C. Removal of Outliers

We can remove outliers in the cluster structure. Clusters that have few data points may represent unusual noises in training data and thus outliers. Hence, clusters with fewer data points than a threshold value are removed. The threshold on the minimum of data points in a cluster can be chosen based on the average number of data points in the produced clusters from CCAS. The thresholds on clusters with different target classes can be different. In this study of applying the robust CCAS to intrusion detection, we set the threshold value to 1.

D. Workflow of the Robust CCAS

The grid-based clustering method provides basic steps in CCAS. The post-processing methods of redistribution, supervised hierarchical clustering and outlier removal can be arranged in any order. Each post-processing method can directly be applied to the produced clusters from another method. Figure 8-2 shows the workflow used in our application of the robust CCAS to intrusion detection. This workflow includes five phases. Phase 1 is necessary because we need to calculate the correlation coefficients in the grid-based clustering method of CCAS and record the value range of each predictor variable with the minimum and maximum values of all the training data points. Phases 2-5 are the major phases of the robust CCAS.

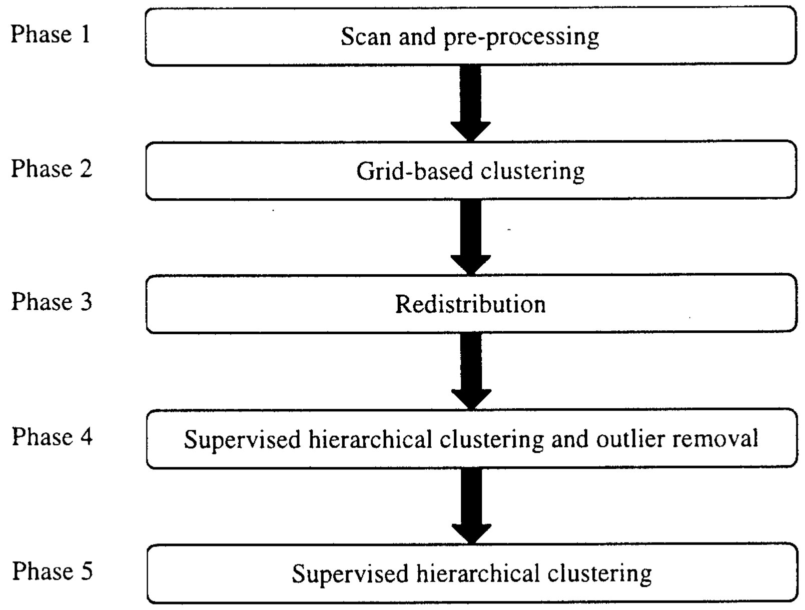


Figure 8-2. Workflow of the robust CCA-S in our application.

1. Phase 1 is the preprocessing step. In this phase, we scan the data points in the training data, calculate the correlation coefficient between each predictor and the target variable, and record the value range for each predictor variable determined by the minimum and maximum values of all the data points.
2. Phase 2 is the grid-based clustering process. We apply the supervised non-hierarchical clustering algorithm to incrementally incorporate all the data points in the training data into clusters.
3. In phase 3, we use the output clusters from phase 2 as the seed clusters to redistribute the training data points to get a more accurate cluster structure.
4. In phase 4, we first apply the supervised hierarchical clustering algorithm. After this, outliers become more obvious. Then we remove possible outliers according to the thresholds set by users.
5. In phase 5, we apply the supervised hierarchical clustering algorithm to the produced clusters from Phase 4 after removing outliers for the second time to get a more compact and accurate cluster structure.

There are differences in the classification stage after each of the major phases of the robust CCAS. We may use all the produced clusters in calculating the target value of a new data point. However, using all the produced clusters is not appropriate when we consider the grid-based clustering method in which grid cells limit the formation of clusters. Therefore, in phases 2 and 3, we use all the clusters in the grid cell of a data point as the nearest neighbors to calculate the target value of the data point. If there is no cluster in this grid cell, the target value of this data point is the same as the nearest cluster to this data point in the entire data space. Grid cells play no role in phases 4 and 5. Hence, we use all the produced clusters in the entire data space to calculate the target value of a new data point.

E. Incremental Learning Ability

As shown in [8-19], CCA-S supports the incremental update of the clusters with new training data. Among the post-processing methods in the robust CCAS, the redistribution method uses the same steps in CCAS but starts with the seed clusters, and thus this method also supports the incremental learning ability. The outlier removal method does not involve incremental learning. Only the supervised hierarchical clustering method needs to be given a special consideration because this method combines original clusters into larger clusters and thus changes the original cluster structure. Hence, to support the incremental learning ability, we need to keep the cluster structure after phase 3.

F. Computation Complexity

Let the number of output clusters after each phase be M. For the first phase of scanning the data points in the training data, the computation complexity is $O(pN)$. For grid-based clustering in phases 2 and 3, the upper bound on the computation cost is $O(pNM)$ if we search the produced cluster structure sequentially. However, we can employ efficient storage and management of the produced clusters. Using the technique such as one in [8-18], this computation cost can be improved to almost linear to $O(pN)$. For the supervised hierarchical clustering method in phases 4 and 5, the computation cost is bounded by a constant that depends on the number of input clusters. The computation of pair-wise distances in the single linkage method of hierarchical clustering is $O(\frac{M(M-1)}{2})$. However, since this supervised hierarchical clustering algorithm allows only clusters with the same class label to be combined, many pairwise distances will need to be computed. With an efficient algorithm in our application, the computation cost in our testing experiments is very low. The computation cost of removing outliers in phase 4 is $O(M)$. The computation cost of classifying a data point is $O(pM)$. Again this computation cost can be reduced to almost $O(p)$ if we use efficient techniques to store and search the cluster structure.

IV. APPLICATION TO INTRUSION DETECTION

The robust CCA-S is tested using a large set of computer audit data for intrusion detection to evaluate detection performance and robustness.

A. Data Representation

Computer audit data used in this application consist of audit records for a sequence of audit events from a host machine with a Solaris operating system. The Basic Security Module (BSM) in the Solaris operating system generates computer audit data. There is an audit record for each audit event. Each audit record has a number of attributes, including event type, user ID, process ID, command type, time, remote IP address, and so on. The target class of an audit event is 1 if the event is from an attack, and 0 otherwise. The target class of each audit event is provided for computer audit data used for training. We also know the true target class of each audit event used for testing. However, we use the robust CCAS to predict the target value of each audit event in the testing data set, and then compare the predicted target value with the true target class of each event for evaluating detection performance of the robust CCAS.

In this study, we use only the information of event type from each audit record because existing work on intrusion detection [8-1 – 8-11] has demonstrated the effectiveness of this data field in intrusion detection. There are in total 284 different types of audit events in this Solaris operating system. We use 284 predictor variables to represent the frequencies of the 284 event types respectively that occur in the recent past of the current audit event. Given a stream of audit events, we use an exponentially weighted moving average (EWMA) technique [8-29] to obtain a smoothed frequency distribution of event types in the recent past of the current audit event. Considering the t^{th} event as the current event, we use the following formula to compute the smoothed frequency distribution of the 284 event types in the recent past of the t^{th} event:

$$\begin{cases} X_i(t) = \lambda \times 1 + (1-\lambda) \times X_i(t-1) \text{ if the current event is the ith event type} \\ X_i(t) = \lambda \times 0 + (1-\lambda) \times X_i(t-1) \text{ if the current event is not the ith event type} \end{cases} \quad (8\text{-}7)$$

where $X_i(t)$ is the smoothed observation value of the i^{th} predictor variable for the current event, and λ is a smoothing constant which determines the decay rate.

The value of each predictor variable captures not only the information of the current event, but also the information of the recent events. With this EWMA method, we add the time characteristic into the values of the predictor variables to represent the frequency distribution of event types in the recent past of the current event. In our study, we initialize $X_i(0)$ to be 0 for i = 1,...,284. We let λ be 0.3—a common value for the smoothing constant [8-29]. Hence, for each event in the training and testing data set, we obtain a vector of (X_1,..., X_{284}).

B. Training and Testing Data

We use the MIT Lincoln Laboratory's 2000 DARPA Intrusion Detection Evaluation Data (http://ideval.ll.mit.edu/) to create our training and testing data. The data are generated in a local-area network (LAN) at the MIT Lincoln Laboratory by simulating activities in a typical U.S. Air Force LAN. The data include several phases of a distributed denial of service (DDoS) attack. This attack scenario is carried out over multiple network sessions. These sessions have been grouped into 5 attack phases, over the course of which the attacker probes, breaks in via some Solaris vulnerability, installs trojan DDoS software, and finally launches a DDoS attack. There are 15 normal sessions and 7 attack sessions in the data stream from the host machine called "Mill," and 63 normal and 4 attack sessions in the other data stream from host machine "Pascal". Each of the two event streams has over a hundred thousand audit records. We use the data from "Pascal" as the training data and the data from "Mill" as the testing data.

To examine the robustness of CCA-S and its extension, we use two different input orders of the training data. In the 2000 DARPA Intrusion Detection Evaluation Data, the attack sessions are mixed with normal sessions. In addition to the original input order, we create a reversed input order by reversing the order of sessions while maintaining the order of audit events in each session. We refer to the original and reversed input orders as input orders 1 and 2 respectively. We also test the sensitivity of CCA-S and its extension to the grid parameter using input order 1. In our study we choose an equal number of grid intervals for each dimension. We use 6 and 11 intervals to build grid cells for input order 1 (see Table 8-2). For input order 2, only 11 intervals are used to build grid cells.

Table 8-2. The number of grid intervals to set up grid cells.

		Input order	
		Original (1)	Reversed (2)
Number of	11	✓	✓
grid intervals	6	✓	

V. RESULT ANALYSIS

Two types of errors occur when using an intrusion detection algorithm. A false positive occurs when the algorithm predicts an event as intrusive when it is in fact normal. A false negative (or a false alarm) occurs when a truly intrusive event occurs but the algorithm allows it to pass as a normal event.

In this study the target value of each audit record in the testing data falls in [0, 1] and indicates the intrusiveness of the corresponding audit event. Given a signal threshold, we signal an event as intrusive if the target value is greater than or equal to this signal threshold; otherwise we claim the event as normal. A signal on a truly normal event is a false alarm. A signal on a truly intrusive event is a hit. A false alarm rate is the ratio of the number of signals on all the normal events in the testing data to the total number of all the normal events in the testing data. A hit rate is the ratio of the number of signals on all the intrusive events in the testing data to the total number of all the intrusive events in the testing data. Hence, given a signal threshold, we obtain a pair of the false alarm rate and the hit rate. We then plot a ROC (Receiver Operating Characteristics) curve using pairs of false alarm rate and hit rate that are generated by varying the value of the signal threshold. In a ROC curve, *X*-axis represents the false alarm rate while *Y*-axis represents the hit rate. The closer the curve is to the top-left corner (100% hit rate and 0% false alarm rate) of the plot, the better the detection performance. Such a ROC analysis is based on the target value of each audit event in the testing data for evaluating detection performance on individual audit events. Hence, it is called the event-based ROC. However, it is not appropriate to detect intrusions based on individual events, because the same audit event may be common in both normal and intrusive activities. It is important to examine aggregate sets of individual events.

In this study we evaluate detection performance on sessions and perform session-based ROC analysis by obtaining a session signal ratio for each session in the testing data. There are usually many audit events in a session. The session-based ROC analysis includes the following steps:

1. Calculate a signal threshold to determine if we should signal an individual event based on its predicted target value. There are several ways to determine this signal threshold. In our study we calculate the average μ and the standard deviation σ of predicted target values for all the normal events in the training data. The signal threshold is set as $\mu + a\sigma$, because we are mainly interested in detecting a large predicted target value close to 1 which indicates an possible intrusion, where a is a coefficient to adjust the signal threshold.
2. Compute a "session table" containing a session signal ratio for each session. For each session we calculate the ratio of the number of signals to the total number of events in the session as the session signal ratio for a given signal threshold.
3. Plot the ROC curve based on the session signal ratios of all the sessions in the testing data.

We expect that session signal ratios for normal sessions are lower than session signal ratios for intrusive sessions. In this study, we experiment with various values of a, and it appears that small a values of 0.5, 0.6, and 0.7 produce the best performance.

Figure 8-3 shows the ROC analysis of the testing results from phases 2, 3, 4, and 5 of the robust CCA-S using 11 grid intervals and input order 1 of the training data. The detection performance of the cluster structure from phase 2 or phase 3 is not satisfactory. The grid-based clustering method and the post-processing method of redistribution in phases 2 and 3 respectively detect only about 60% of the attack sessions when no false alarms are produced on normal sessions. This indicates that 5 of the 7 attack sessions are detected. After phase 4 using the supervised hierarchical clustering method and the outlier removal method, we obtain the 100% detection rate with no false alarms using two signal thresholds. After phase 5 using the supervised hierarchical clustering method, all the three signal thresholds produce the 100% detection rate and the 0% false alarm rate. In this testing data set, there are two attack sessions, each of which consists of only one audit event. Attack sessions with few audit events are very difficult to detect. The robust CCA-S detects both attack sessions and demonstrates good performance.

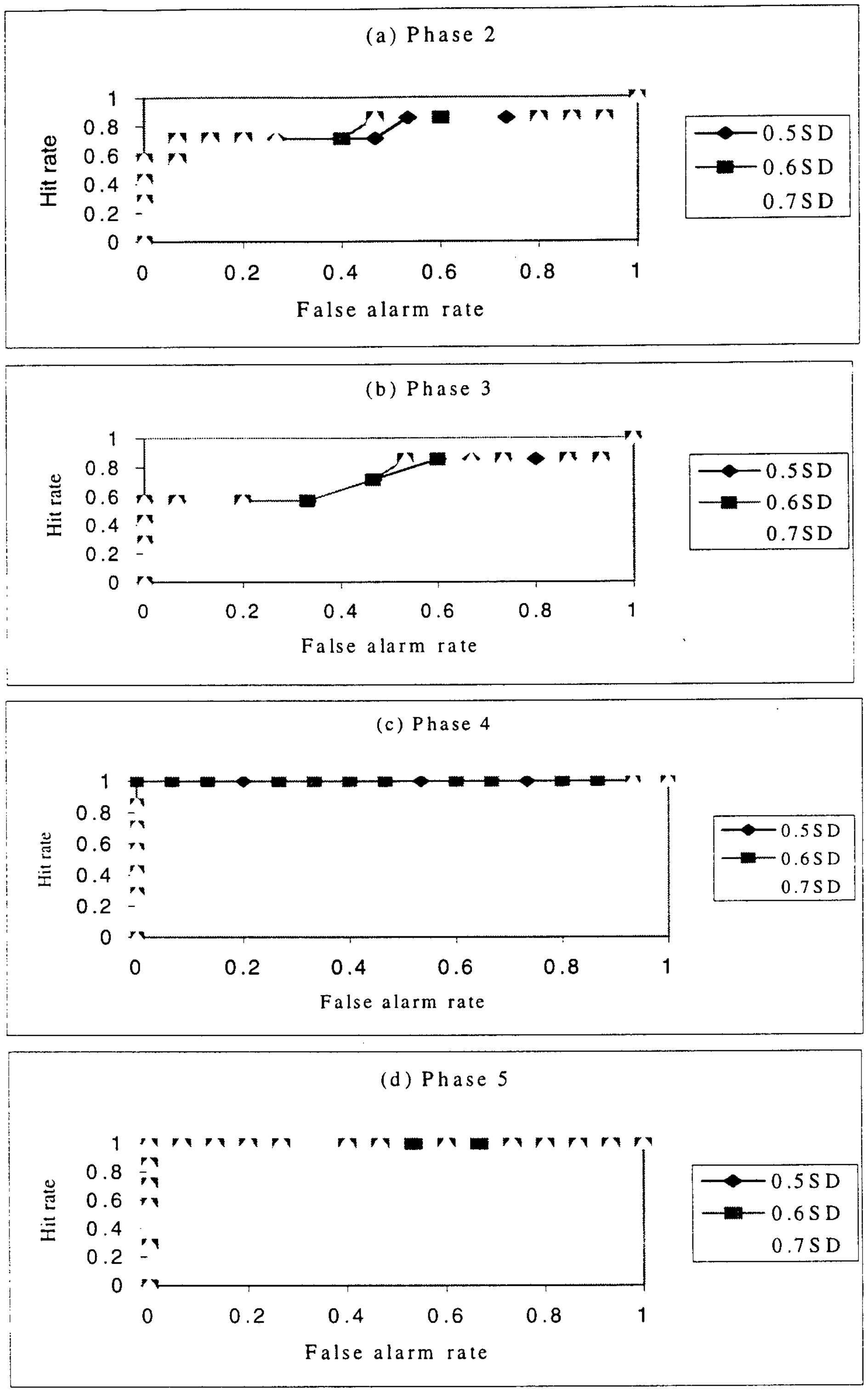
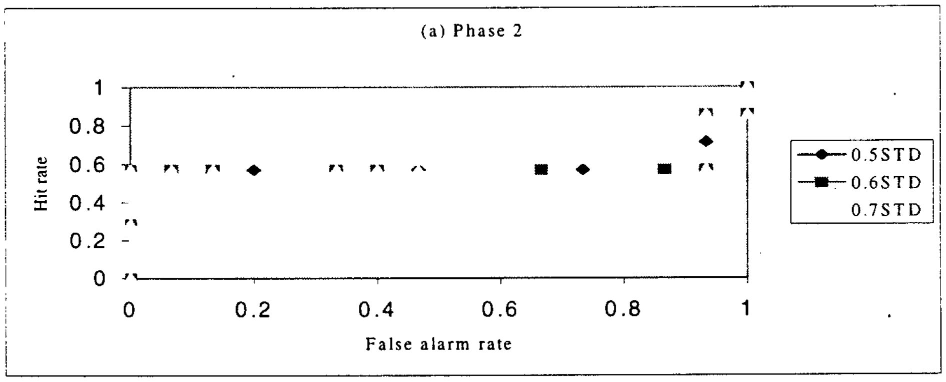
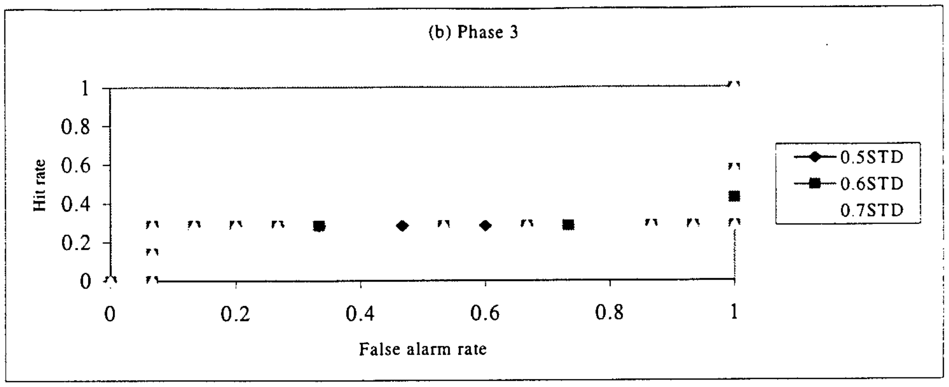
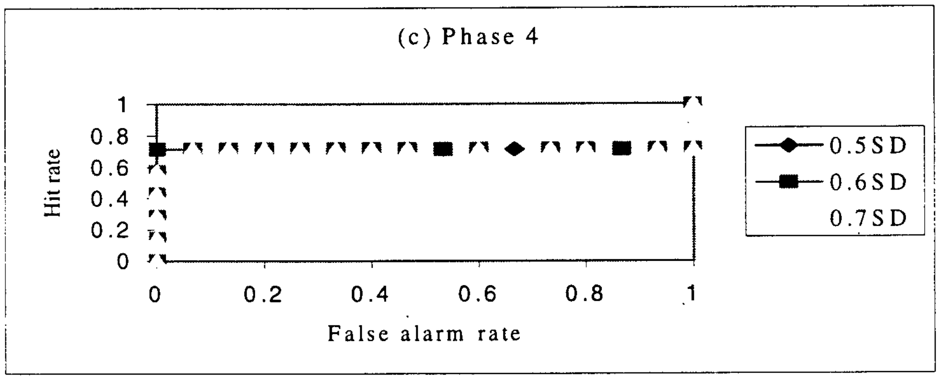


Figure 8-3. ROC analysis for the robust CCA-S using 11 grid intervals and input order 1.

To examine the robustness of CCAS and its extension to input order of training data, we perform ROC analysis for input order 2 of the training data (see Figure 8-4). The number of grid intervals to build grid cells is still 11. From the ROC curves in Figure 8-4, we can see that the input order of the training data has impact on the grid-based clustering method in phase 2 and the redistribution method of post-processing in phase 3, because the ROC curves for this input order are worse than those for input order 1, especially in phase 3. After the supervised hierarchical clustering method and the outlier removal method of post-processing in phase 4, the detection performance becomes a little better than that in both phases 2 and 3. One signal threshold produces the best performance with a detection rate of a little above 70% and no false alarms. After the supervised hierarchical clustering method in phase 5, we again obtain good detection performance with two signal thresholds producing the 100% hit rate and the 0% false alarm rate. Hence, the additional steps in the robust CCAS have made detection performance of CCAS robust to the different input orders of the training data.







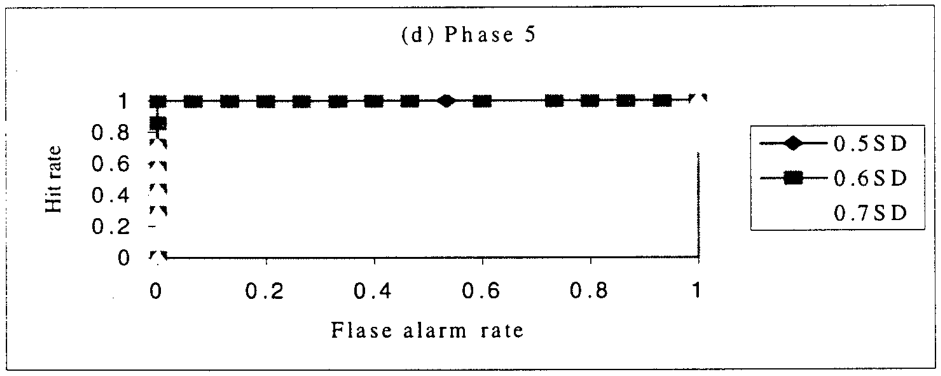
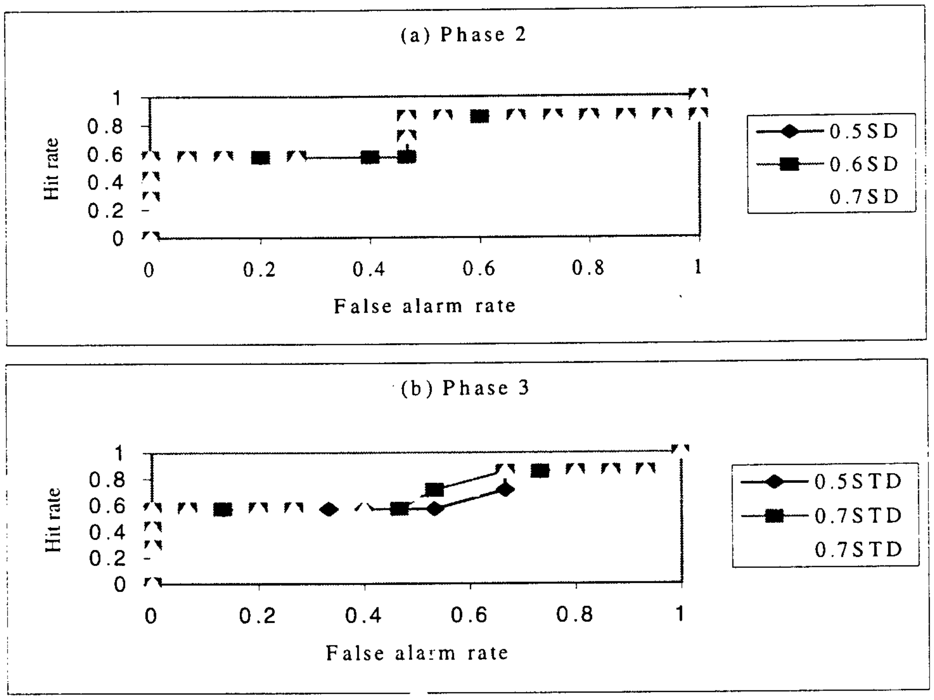
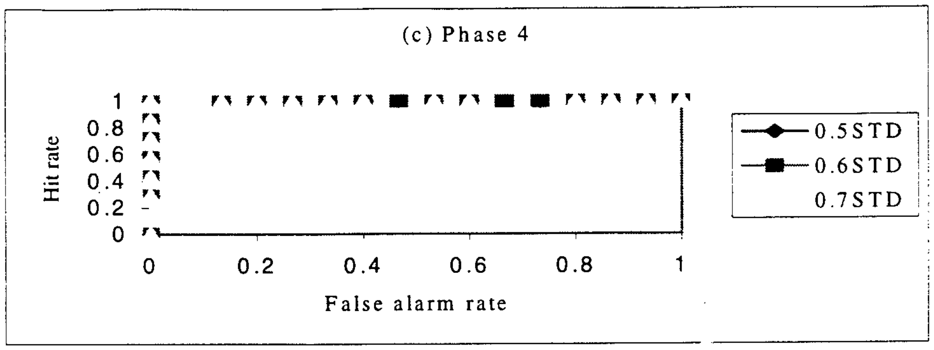


Figure 8-4. ROC analysis for the robust CCA-S using 11 grid intervals and input order 2.

We also test the sensitivity of CCA-S and its extension to the grid parameter by obtaining detection performance of CCAS and its extension using 6 grid intervals in each dimension to building grid cells. This is tested only for input order 1 for training data to see any difference between 11 grid intervals and 6 grid intervals and thus the sensitivity of CCAS and its extension to this grid parameter. Figure 8-5 shows the ROC analysis for the 6 grid intervals which is compared with the ROC analysis in Figure 8-3 for the 11 grid intervals. We can see that there is a slight difference between detection performance of 6 grid intervals and 11 grid intervals after phase 2 for the same input order. However, after phase 5 the 6 grid intervals produce the same detection performance as the 11 grid intervals, indicating that the robust CCAS is not sensitive to the grid parameter.





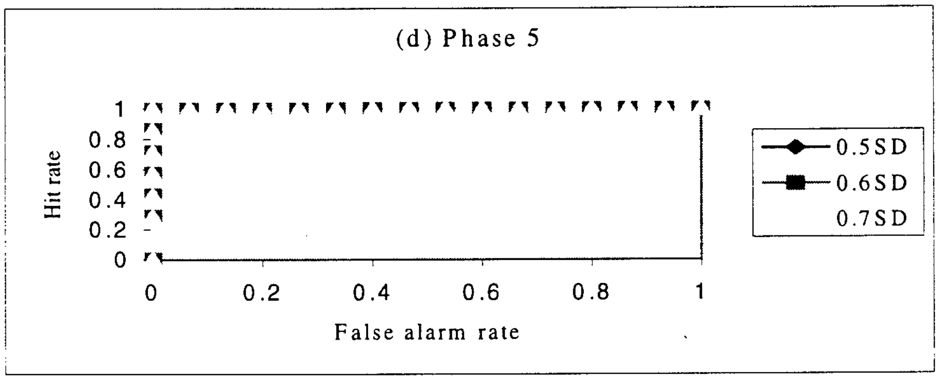


Figure 8-5. ROC analysis for the robust CCA-S using 6 grid intervals and input order 1.

It is expected that finer grid cells may produce better performance since they lead to more clusters and thus allow important patterns of data points to be captured. If the number of grid intervals is too small, natural clusters of the same target class may merge into larger clusters due to an unusual input order of training data points in the incremental clustering procedure, consequently resulting in unsatisfactory classification performance. However, too many grid intervals force natural clusters to be separated into smaller clusters. In some special cases, this may produce a poor cluster structure even after Phases 4 and 5 and thus poor classification performance. Another shortcoming of a small size of grid interval is high computation and storage cost. The robust CCA-S provides robustness to the grid parameter to some extent.

VI. CONCLUSION

In this chapter we present the robust CCAS—a robust, scalable data mining algorithm with the incremental learning ability. The robust CCAS adds the post-processing steps to the basic steps of CCAS, including the redistribution of training data points, the supervised hierarchical clustering of clusters, and the removal of outliers which are applied to the produced clusters from CCAS. The robust CCAS is tested using a large set of computer audit data for intrusion detection. The testing results show that the robust CCAS makes significant improvement in detection performance and robustness to input order of training data and noises. The application of the robust CCAS is not limited to intrusion detection. It can also be applied to other data mining problems for classification.

REFERENCES

[8-1] D. E. Denning. "An Intrusion-detection Model," *IEEE Transactions on Software Engineering*, SE-13(2), pp. 222-232, February 1987.

[8-2] H. Debar, M. Dacier, A. Wespi, "Towards a Taxonomy of Intrusion-detection Systems," *Computer Networks*, vol. 31, pp. 805-822, 1999.

[8-3] R. G. Bace, *Intrusion Detection*, Macmillan Technical Publishing, 2000.

[8-4] R. Lippmann, D. Fried, I. Graf, J. Haines, K., Kendall, D. McClung, D. Weber, S. Webster, D. Wyschogrod, R. Cunningham, and M. Zissman, "Evaluating Intrusion Detection Systems: The 1998 DARPA Off-line Intrusion Detection Evaluation," *In Proceedings of the DARPA Information Survivability Conference and Exposition,* Los Alamitos, CA: IEE Computer Society, pp. 12-26, January 2000.

[8-5] J. McHugh, A. Christie, and J. Allen, "Defending Yourself: The Role of Intrusion Detection Systems," IEEE Software September/October 2000. pp. 42–51

[8-6] N. Ye, X. Li, Q. Chen, S. M. Emran, and M. Xu, "Probabilistic Techniques for Intrusion Detection Based on Computer Audit Data," *IEEE Transactions on Systems, Man, and Cybernetics,* Vol. 31, No. 4, 2001, pp. 266-274.

[8-7] N. Ye, J. Giordano, and J. Feldman, "A Process Control Approach to Cyber Attack Detection," *Communications of the ACM,* Vol. 44, No. 8, 2001, pp. 76-82.

[8-8] S. M. Emran and N. Ye, "A System Architecture for Computer Intrusion Detection," *Information, Knowledge, Systems Management,* Vol. 2, No. 3, 2001, pp. 271-290.

[8-9] T. Escamilla, *Intrusion Detection: Network Security beyond the Firewall*, New York: John Wiley & Sons, 1998.

[8-10] N. Ye, X. Li, and S. M. Emran, "Decision Trees for Signature Recognition and State Classification," In *Proceedings of the First IEEE SMC Information Assurance and Security Workshop*, 2000, pp. 189-194.

[8-11] X. Li, and N. Ye, "Decision Tree Classifiers for Computer Intrusion Detection," *Journal of Parallel and Distributed Computing Practices,* accepted.

[8-12] W. Lee, S. J. Stolfo, K. Mok, "A Data Mining Framework for Building Intrusion Detection Models," In *Proceedings of 1999 IEEE Symposium on Security and Privac,* 1999, pp. 120-132.

[8-13] C. Sinclair, L. Pierce, S. Matzner, "An Application of Machine Learning to Network Intrusion Detection," In *Proceedings of 15th Annual Computer Security Applications Conference (ACSAC '99),* 1999, pp. 371-377.

[8-14] D. Endler, "Intrusion Ddetection: Applying Machine Learning to Solaris Audit Data," In *Proceedings of 14th Annual Computer Security Applications Conference*, 1998, pp. 268–279.

[8-15] A. Valdes, K. Skinner, "Adaptive, Model-based Monitoring for Cyber Attack Detection," In Proceedings of *Recent Advances in Intrusion Detection (RAID) 2000,* 2000; Toulouse, France.

[8-16] N. Ye, and X. Li, "A Supervised, Incremental Learning Algorithm for Classification Problems," *Computers & Industrial Engineering Journal,* accepted.

[8-17] N. Ye, and X. Li, "A Supervised Clustering and Classification Algorithm for Computer Intrusion Detection," *IEEE Transactions on Systems, Man, and Cybernetics,* submitted.

[8-18] N. Ye, and X. Li, "A Scalable Clustering Technique for Intrusion Signature Recognition," In *Proceedings of the Second IEEE SMC Information Assurance Workshop*, 2001, pp. 1-4.

[8-19] X. Li, and N. Ye, "Grid- and Dummy-cluster-based Learning of Normal and Intrusive Clusters for Computer Intrusion Detection," *Quality and Reliability Engineering International,* submitted.

[8-20] C. M. Procopiuc, *Clustering Problems and Their Applications (a Survey),* 1997; Department of Computer Science, Duke University.

[8-21] A. K. Jain, R. C. Dubes, *Algorithms for Clustering Data*, 1988; Prentice Hall.

[8-22] T. Zhang, *Data Clustering For Very Large Datasets Plus Applications,* 1997; Ph.D. Thesis, Department of Computer Science, University of Wisconsin - Madison.

[8-23] M. Ester, H. P. Kriegel, J. Sander, M. Wimmer, X. Xu, "Incremental Clustering for Mining in a data Warehousing Environment," *Proceedings of 24th VLDB Conference,* 1998; New York, USA.

[8-24] G. Sheikholeslami, S. Chatterjee, A. Zhang, "WaveCluster: A Multi-Resolution Clustering Approach for Very Large Spatial Databases," *Proceedings of 24th VLDB Conference,* 1998; New York, USA.

[8-25] S. G. Harsha, A. Choudhary, *Parallel Subspace Clustering for Very Large Data Sets,* Techinical report, CPDC-TR-9906-010, 1999; Northwestern University.

[8-26] C. Huang, Q. Bi, R. Stiles, and R. Harris, "Fast Search Equivalent Encoding Algorithms for Image Compression Using Vector Quantization," *IEEE Trans. on Image Processing,* vol. 1, no. 3, pp. 413-416.

[8-27] V. Cherkassky, F. Mulier, *Learning from Data,* 1998; John Wiley & Sons.

[8-28] T. Mitchell, *Machine Learning*, 1997; WCB/McGraw-Hill.

[8-29] T. P. Ryan, *Statistical Methods for Quality Improvement,* 1989; New York: John Wiley & Sons.

Information Dynamics and Emergent Behavior of Heterogeneous-Agent Systems

GRANT NO.:	F49620-99-1-0014
PI:	Nong Ye
SPONSOR:	AFOSR/DARPA
REPORTING PERIOD:	January 1, 2001 – December 31, 2001

Table of Contents

List of Figure Captions

List of Table Captions

ABSTRACT

This project presents an effort to establish theories and techniques of modeling, analyzing and controlling information dynamics and emergent behavior of heterogeneous-agent systems and demonstrate the application of these theories and techniques to heterogeneous-agent systems making up information supply networks. A heterogeneous-agent system is a Complex Adaptive System (CAS) involving a large collection of semi-autonomous agents. According to Complexity Theory for CAS, the aggregate system behavior of heterogeneous agents and their inter-connections emerges from the evolving local interactions of agents in a dynamically changing environment. A CAS best organizes from the bottom up through self-organization. We base the modeling of heterogeneous-agent systems on the innovative integration of three modeling paradigms: agent-based modeling, control-theoretic modeling, and stochastic discrete-event modeling. The model represents the physical, information and knowledge elements of each agent, which includes: fitness function, tagging, building blocks, internal model, dynamic resources and processes under the control of the agent, as well as the interaction of the agent with other agents and the environment. The analysis of heterogeneous-agent systems and their emergent behavior is based on the innovative application of non-linear time-series analysis techniques from chaos theory and nonlinear dynamical systems theory as well as multivariate statistical techniques to detect emergent states and temporal patterns. We base the control of heterogeneous-agent systems on the innovative bottom-up self-organization approach to coping with desirable and undesirable emergent states of heterogeneous-agent systems for system stability and robustness. This project investigates the emergent behavior of information supply network systems (e.g., military information supply networks for information distribution and fusion in command and control – C2, and commercial information supply networks for e-commerce). In this project, we build and use the simulation model of an information supply network system to collect behavioral data of the system under experimental conditions involving various inter-agent control methods and different network environments. This project also develops and applies analytical techniques to the simulation data for understanding

emergent system behavior under various conditions and gaining insights into system design and control. The simulation experiments look into the following factors:

1) Inter-agent control methods for different degrees of agent autonomy and agent coordination, such as sharing of state information among agents, conformance to a common fitness function by agents, enforcement of the same internal model among agents and so forth, and

2) System structures determined by types and density of inter-connections among agents through material, information and knowledge flows.

INTRODUCTION

A supply network enterprise is a network of organizations coupled for the purpose of providing value to customers. A supply network enterprise consists of a focal organization and the network of firms that transact with it in the form of physical goods and services as well as information. These connections are then extended iteratively (i.e., connecting the supplier's suppliers and customers and the customer's customers and other suppliers). An enterprise potentially transcends multiple industries and markets. Each organization is an agent acting on its own self-interest, but subject to constraints from its contracting relationships with other agents and from the environment. Agents are heterogeneous in their role and other features in a supply network enterprise. A complex network of suppliers, manufacturers, distributors, transporters, retailers, and customers – a supply network enterprise – is common. The complexity of interrelationships arises from not only material flows but also information flows throughout the supply network enterprise. Little exists now in understanding the emergent behavior of supply network enterprises, designing their structures, and optimizing their operation.

A supply network enterprise is a heterogeneous-agent system – a Complex Adaptive System (CAS) – involving a large collection of semi-autonomous agents that may play different roles in the system. According to Complexity Theory for CAS, the aggregate system behavior of heterogeneous agents and their inter-connections emerges form the evolving, local interactions of agents in a dynamically changing environment.

Advances in information technology have led us to a new era of business organization and market structure. Information technology gives a focal organization in a supply network enterprise an unprecedented capability to communicate, coordinate, and even control its suppliers, distributors, transporters, retailers, customers and itself. For example, the focal organization may attempt to improve the delivery time, product quality and cost by obtaining the information of inventory and process quality from suppliers, the information of product sales from retailers, and even coordinating and controlling the business process throughout the supply network via Enterprise Resource Planning (ERP) software. The focal organization can even interact with customers directly to provide products and services and in turn, consumers can benefit by obtaining objective comparative information about their desired purchase.

Despite potential changes that can be introduced by information technology, it is still not clear what kinds of changes will produce the desirable emergent behavior of a supply network enterprise, and thus should be adopted. For example, the following questions remain to be answered:

- Should a focal organization exert tight control over its suppliers by monitoring and controlling the product development and production process of its suppliers and the inventory management process of its distributors, which is made possible by information technology, or should the focal organization leave a high degree of autonomy to its suppliers and distributors for innovation and flexibility?
- What kind of suppliers should a focal organization choose to work with, a few suppliers or a large number of suppliers, suppliers which commit all of their resources to support the focal organization for a long period of time, or suppliers which have more freedom to remain or leave the supply network enterprise dynamically?
- How should a focal organization deal with other agents in the supply network enterprise when facing an unstable market with many changes, versus a stable market?
- Which inter-agent control methods and which supply network structures, under which nature of internal dynamics and eternal changes, leads to the desirable emergent behavior of a supply network enterprise?

The long-term goal of this research work is to establish theories and techniques of modeling, analyzing and controlling heterogeneous-agent systems and their emergent behavior, and to demonstrate the application of these theories and techniques to supply network enterprises in e-business. The work on the simulation modeling of heterogeneous-agent systems will focus on a coherent integration of three modeling paradigms: agent-based modeling, control-theoretic modeling and stochastic process modeling, to represent essential features of agents such as the role, internal model, building blocks, fitness function, physical and information flows, aggregation, diversity and non-linearity. The work on the analysis of heterogeneous-agent systems and their emergent behavior will focus on non-linear time-series analysis techniques from chaos and dynamical systems theories and multivariate statistical techniques to detect emergent states and temporal patterns. The work on the control of heterogeneous-agent systems will focus on the bottom-up self-synchronization approach to handling desirable and undesirable emergent states of heterogeneous-agent systems for system stability and robustness according to Complexity Theory for CAS.

This project report represents the objectives of the initial one-year effort:

1) Build and investigate a simulation model of a supply network enterprise – a heterogeneous-agent system – under experimental conditions involving various inter-agent control methods, internal dynamics, and external changes, and

2) Develop and apply analytical techniques to the simulation data for understanding the emergent system behavior under various conditions and gaining insights into system design and control.

Findings from this effort will provide insights into supply network management (e.g., selection and management of suppliers) as well as insights into the emergent behavior of heterogeneous-agent systems in general. This effort will also demonstrate the feasibility and validity of our simulation modeling approach and analytical methodology for the understanding of heterogeneous-agent systems.

After this one-year effort, we plan to investigate control strategies handling desirable and undesirable emergent states of CAS for the performance, stability, and robustness of CAS.

Chapter 1: Complex Adaptive Systems

Recent efforts have shown that due to the complex, dynamic nature of a supply network, it is not enough to model the network as a mere system. More appropriately, a supply network should be modeled as a Complex Adaptive System (CAS) [1]. Therefore, in our experiments, we model the supply chain enterprise as a heterogeneous-agent CAS.

1-1 Complex Adaptive System Theory

A heterogeneous-agent system is a CAS involving a large collection of semi-autonomous agents that may play different roles in the system. Within each agent is a set of states and behaviors that can be used to describe the agent. Each agent works independently to increase the level of fitness achieved by the agent as well as the surrounding regional or global network of which it is a member. Agents within the CAS are allowed measurable degrees of freedom to behave in a semi-autonomous fashion. The degree of freedom that is afforded an agent is determined by the dimensionality of the CAS. [1] By reducing the dimensionality (through control techniques) or increasing the dimensionality by simply allowing the agents a higher degree of autonomy, we can decrease or increase the level of stochastic behavior present in the CAS respectively. This technique allows us to observe forced behavior under tight control, versus the emergent behavior of a less restricted network.

A key concept in CAS Theory is the connectivity of the agents within the CAS. The number of connections within the system, as well as the characteristics (including level of communication) of each connection dictates the dynamics of the system's communication capabilities. These connections between agents form multiple relationships of varying degrees. The interrelations formed by connectivity are indicative of the network's potential to "engage in global communication from within" [1].

An interactive relationship is present between a CAS and the external environment in which it exists [1]. According to Complexity Theory for CAS, the aggregate system behavior of heterogeneous agents and their inter-connections emerges from the evolving, local interactions of agents in a dynamically changing environment. Therefore, the observed behavior in a CAS is not governed by a single entity, but by the simultaneous actions of the agents within the system, as well as the co-evolution of both the system itself and its environment [1]. A CAS is thus considered a self-organizing entity with observable emergent behavior.

1-2 Supply Network Enterprise

A supply network enterprise is a network of organizations coupled for the purpose of providing value to a customer. A supply network enterprise consists of a focal organization and the network of firms that transact with it in the form of physical goods and services as well as information. These connections are then extended iteratively (i.e., connecting the supplier's suppliers and customers and the customer's customers and other suppliers). An enterprise potentially transcends multiple industries and markets. Agents in a supply network enterprise are subject to varying degrees of connectivity with other agents [1]. This structure determines the availability of information flow through the network.

Each organization is an agent acting on its own self-interest, but subject to constraints from its contracting relationships with other agents and from the environments. Agents are heterogeneous in their role and other features in a supply network enterprise. Complex networks of suppliers, manufacturers, distributors, retailers, and customers – an enterprise – are commonplace in many industries, including E-Commerce. The complexity of interrelationships arises from not only material flows, but also information flows throughout the enterprise. Little exists now in understanding the emergent behavior of supply network enterprises, designing their structures, and optimizing their operation.

The theory of Complex Adaptive Systems seems to fit naturally into the description of a supply network enterprise. Due to this observation, we model our supply network enterprise as a CAS.

Chapter 2: Modeling a Supply Network Enterprise

The system modeling will be based on a coherent integration of three modeling paradigms: agent-based modeling, control-theoretic modeling, and stochastic process modeling as shown in the following figure. This structure applies the comprehensive cohesion of numerous current techniques used in modeling supply network enterprises. [2-6]

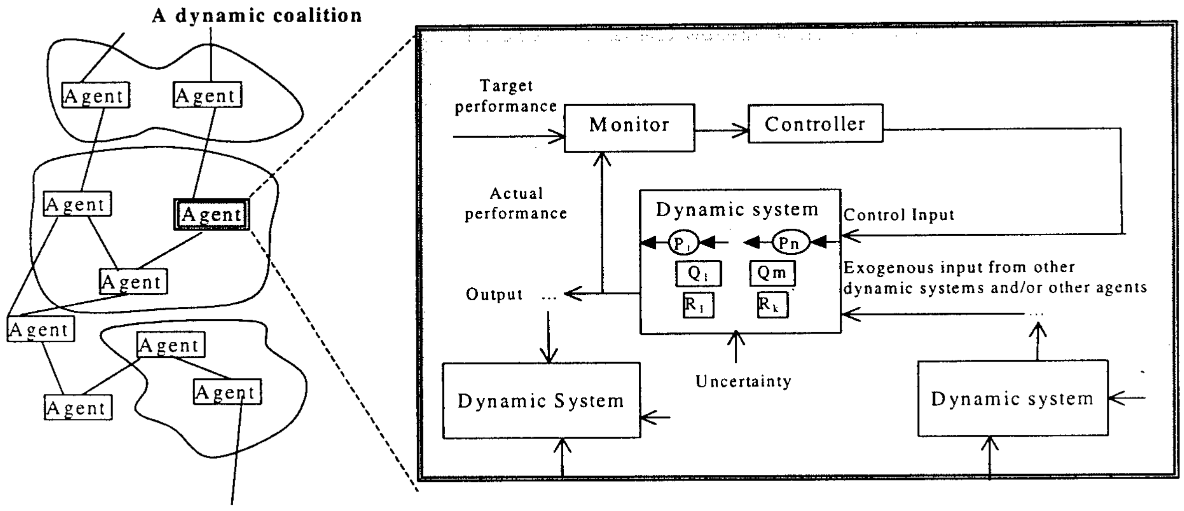


Figure 2-1: Complex adaptive system model of a supply network enterprise.

Control-theoretic modeling based on control theory will be used to specify the proactive, negative-feedback logic of an agent in the controller of the agent for the adaptive, unexpected behavior. Stochastic process modeling based on queuing theory and stochastic process theory will be used to specify dynamic systems and uncertainties in the environment from which agents gain information and on which agents produce impact.

We use these three functions, which are described, in section 2-2.2 in the following sections to describe agent fitness and production calculations:

gap(i) =	*production(i) - demand(i)*
customerGap(i) =	*production(i) - demand(customer)*
endGap(i) =	*production(i) - demand(endCustomer)*

2-1 Agent Design

Each agent will contain the following elements: role, fitness function, internal model, building blocks, dynamic system controlled by the agent, and connectivity with other agents and the environment. At any time, the agent's state can be defined by the state of each of its elements.

2-1.1 Role of Agent

The role of an agent indicates the type of the agent. An agent's type can be a focal organization, supplier, or customer. The agent's role in the system depends on its type. Suppliers are measured by their distance from the focal organization, for example, suppliers that supply directly to the focal organization are 1-tier suppliers, their suppliers are 2-tier suppliers, and so on. For every supplier, the supplier's customer exists at a tier one level above (numerically lower than) the supplier. This structure enables us to observe sub-structures within the network. For example, a supplier, along with its immediate suppliers and customer, forms a regional network that exists within the global supply network enterprise where the supplier acts as the focal organization for the smaller sub-network.

2-1.2 Fitness Function

The fitness function describes the objective of the agent, and the relationship of the objective measure (e.g., profit), with various factors. These factors include: the state of the dynamic system (e.g., the inventory level), the output performance (e.g., number of finished orders, delivery time, and product quality) of the dynamic system, the number of lost orders due to the inability of the agent to quickly accommodate changes and to attract new orders, etc. The agent behaves in a manner as to increase the fitness of the system that the agent belongs to locally, regionally and globally. To the extent that fitness criteria are shared, the aggregate group of agents tends to act in a collaborative fashion.

Each agent maintains this metric for its "local" fitness, as well as the "regional fitness" of the agent along with its immediate suppliers and customer. The following formulas which we use to calculate the fitness levels consider "0" to be optimal fitness. The *gap* measurement is mentioned above, and described in section 2-2.2. The *Δproduction* is simply the change in production from the previous cycle to the current cycle.

$$localFitness = |gap| + |\Delta production|$$

$$regionalFitness = \left(\sum_{i=1}^{r} localFitness(i) \right) / r$$

r = "number of <u>connected</u> customers and suppliers + 1"

The fitness function plays an important part in the evolving internal model. An agent can change its internal model based on its fitness levels, and the current state of the network. Agents monitor their fitness criteria with the use of fitness and gap functions to trigger the bounded control adjustment.

2-1.3 Internal Model

The internal model describes the reactive and proactive logic that the agent follows to determine its behavior. This model acts as the mental model or schema that the agent uses at any given moment to interpret its behavior and its external environments, and to generate the control inputs to the dynamic systems and the interactions with other agents. Hence, the internal model consists of the intra-agent control strategy in the controller and the performance assessment method. A part or the entirety of the internal model can be shared among a group of agents (e.g., shared norms, values, beliefs, and assumptions that make up the agent's internal model) or may be highly individualistic. The internal model is subject to evolution through learning and adaptation.

At any given time, the internal model of the agent can be built up by selecting any combination of building blocks associated with that agent. Decisions can then be made based on negative (achieving the goal set by the focal organization) and positive feedback (deviation from the goal set by the focal organization by having the focal organization accept this new goal). After making decisions, an agent can observe the direct effect of the decision on its fitness levels, and make adjustments accordingly.

The agents in our simulation select rules from the building blocks based on the homogeneity of the network and the level of information sharing between agents. These concepts are described in detail in section 2-2.2.

2-1.4 Building Blocks

The building blocks are sets of control and adaptation rules, algorithms and strategies from which the agent can choose to compose its internal model at any time. These may include control-theoretic algorithms, optimization algorithms, genetic algorithms, heuristic rules, and performance assessment methods. For example, agents are more cooperative under centralized control (top-down command, deterministic planning and re-planning in response to real time events, negative feedback control, proactive logic, the focal organization selects suppliers at all tiers, sets the unit prices and production levels throughout the supply network, service level, type of contract, and resource commitment of each agent). Agents are less cooperative under distributed control (bottom-up synchronization, positive feedback control, distributed decision making, reactive and adaptive logic, offer and counter-offer, accept or reject, all decisions involve only two parties: the upstream agent and the downstream agent). Levels of cooperation apply to issues of price, service level, resource commitment, and the selection and management of suppliers.

The building blocks included in our simulation include control strategies for determining production levels based on current level of homogeneity and information sharing within the network. The control strategies (described below) refer to network types and information sharing levels, which are described in section 2-2.2.

NETWORK TYPE	INFORMATION SHARING LEVEL	IF CONDITION:	SET PRODUCTION TO:
homogeneous	None	|gap| > 15 for 3 cycles	production – avgGap
	Regional	|customerGap| > 15 for 3 cycles	production - avgCustomerGap
	Global	|endGap| > 15 for 3 cycles	production – avgEndGap
semi-hetero & heterogeneous X={1..30} Y={1..6}	None	|gap| > X for Y cycles	production – avgGap
	Regional	|customerGap| > X for Y cycles	production - avgCustomerGap
	Global	|endGap| > X for Y cycles	production – avgEndGap

Table 2-1 Control strategies to set production level

2-1.5 Dynamic System

The dynamic system consists of mainly processes representing orders for finished goods, and resources taking processes and producing finished goods. Many firms satisfy orders by taking finished goods from the inventory, and then filling up the inventory through production using resources. Hence, material flows in the dynamic system may well be represented through the inventory level rather than the process flow of orders on resources. Our model of a supply network enterprises simplifies the representation of process flows on resources by applying the PUSH/PULL method of inventory control where orders are filled using inventory, and inventory is then filled using production. The inability to fill an order due to lack of sufficient inventory is reflected in the agent's fitness function.

2-1.6 Connectivity

The connectivity of the agent with other agents may manifest through material flows, information flows (e.g., the sharing of inventory information), and knowledge flows (e.g., the sharing of the internal model and/or fitness function). Different types and densities of connectivity determine different degrees of inter-agent control. For example, when there exists only material flows between a focal organization and one of its suppliers, the degree of inter-agent control is low. When there also exists information flows and even knowledge flows, the degree of inter-agent control between the focal organization and the supplier is high.

Agents in our simulation experiments are connected to one customer agent, and multiple supplier agents in a tree-like fashion. We present the details of this structure in section 2-2.1.

2-2 Network Design

In our experiments, we investigate different types of networks. Each of these networks is based on the same physical network structure. The variations introduced in this section represent variations on the state of the supply network as opposed to the physical positioning of the agents within the network.

2-2.1 Physical Structure

We choose a real supply network enterprise to determine the structure of our simulation model. This is a fixed network structure using a supply network with twenty-nine agents. The physical structure of the network maintains a hierarchy specified by the focal organization, followed by successive tiers of suppliers. During simulation, demand functions (described in section 2-2.2) determine the initial customer demand to place on the focal organization. This demand then propagates through the network in such a way that the demand placed on a supplier agent is equal to the production level of that agent's customer. This demand/production relationship is formed iteratively from the focal organization down to the farthest supplier tier.

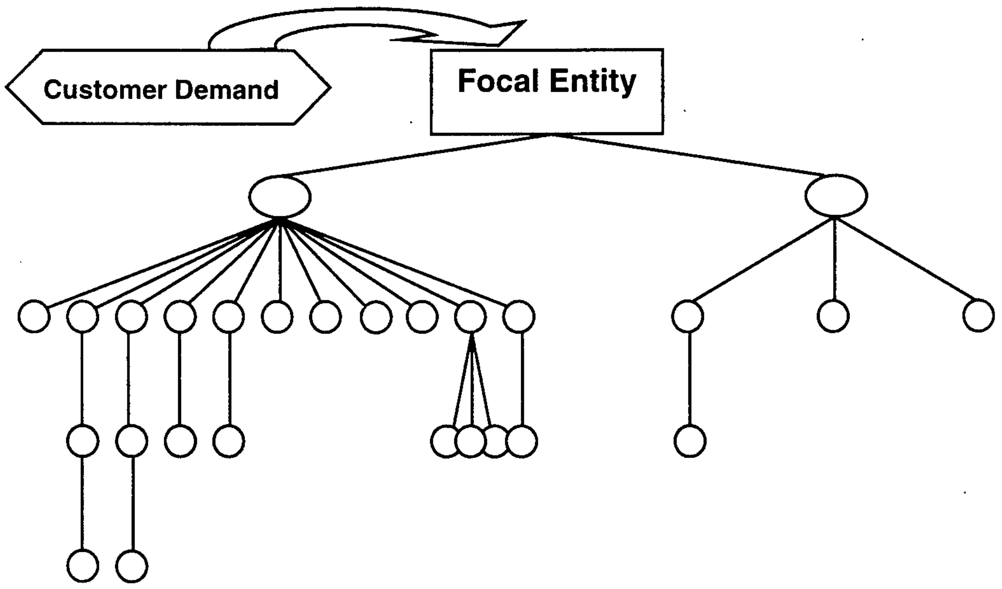


Figure 2-2 The structure of the supply chain enterprise used in the experiments.

The state of the network at any given time is defined by the type of the network (described in section 2-2.2) in conjunction with the network's fitness levels (described in section 2-2.3).

2-2.2 Network Types

At any given time, the state of the supply network enterprise can be given by its type and fitness levels (described in the next section). In this section, we define the type of a network.

There are four factors used to describe the type of a network. These factors are outlined in the following table.

FACTOR	POSSIBLE VALUES
Homogeneity	[homogeneous I semi-hetero I heterogeneous]
Level of Autonomy	[low I high]
Level of Information Sharing	[none I regional I global]
Market Condition	[stable I increasing I decreasing I volatile I seasonal]

Table 2-2 Factors that describe network types.

The homogeneity variable determines the extent to which agents within the system act in a similar fashion. In a homogeneous network, every agent has identical building blocks that it uses to build up its internal model, and each agent has the same initial production level. At the semi-hetero level, agents still have the same initial production level, but now have varying building blocks. The building blocks we use dictate when an agent will trigger a change in its dynamic system, and how much of a change will occur. These building blocks are described in Table 2-1. In the homogeneous system, the agents all modify their production values under the `same circumstances (i.e. gap>15 for 5 cycles). In a semi-hetero network, the values 15 and 5 are replaced by randomly generated values in the ranges {1..30} and {1..6} respectively. The pre-determined randomly generated values used in our experiments are shown in the following table.

AGENT	X	Y	AGENT	X	Y	AGENT	X	Y
0F	6	3	2H	5	3	3D	29	2
1A	15	4	2I	13	3	3E	1	2
1B	6	5	2J	2	3	3F	4	5
2A	10	1	2K	13	3	3G	26	3
2B	8	1	2L	22	4	3H	28	5
2C	28	3	2M	16	4	3I	23	5
2D	10	2	2N	15	1	4A	2	3
2E	30	4	3A	1	6	4B	8	3
2F	3	3	3B	10	2			
2G	25	6	3C	12	6			

Table 2-3 Randomly generated values for heterogeneous network functions.

In this and the following table, the name given to an agent includes its level in the network structure (number), and the order in which it appears in that level (letter). The X and Y columns correspond to the X and Y values of the production functions shown in section 2-1.4. In a heterogeneous network, the building blocks remain varied, and each agent now has a unique, randomly generated, initial production level in the range {500-1,500} as shown in the table below.

AGENT	PRODUCTION	AGENT	PRODUCTION	AGENT	PRODUCTION
0F	1262	2H	743	3D	1016
1A	998	2I	1350	3E	718
1B	1377	2J	1230	3F	937
2A	745	2K	792	3G	1311
2B	1306	2L	1336	3H	841
2C	558	2M	1462	3I	1063
2D	703	2N	905	4A	886
2E	562	3A	1407	4B	503
2F	1428	3B	540		
2G	960	3C	1039		

Table 2-4 Randomly generated initial production values

The level of autonomy determines how much freedom is given to individual agents within the system to make their own decisions. Under high autonomy, the focal organization sets all the production values throughout the network using a production function based on the type of network. In low autonomy, each agent uses the production functions to determine its own production level, which is also based on the type of network. Therefore, the level of autonomy simply determines whether the network is operating under centralized or distributed control.

The level of information sharing within the network dictates how much a supplier knows about its customer. In our simulation experiments, we model the information sharing by giving the agents varying levels of access to upstream agents' demands. The three levels of information sharing are reflected in the three gap functions mentioned at the beginning of this chapter and restated here for convenience:

INFORMATION SHARING LEVEL	IMPLICATIONS	RESULTING GAP FUNCTION
None	Agent is only aware of its own demand	*production(i) - demand(i)*
Regional	Agent is aware of its demand and its immediate customers demand	*production(i) - demand(customer)*
Global	Agent is aware of its demand, its immediate customers demand, and the end customer demand imposed on focal organization.	*production(i) - demand(endCustomer)*

Table 2-5 Result of information sharing on agent knowledge (the gap functions).

These gap functions are a reflection of the knowledge an agent possesses of its environment. At each level, the agent has access to the information available at the level it is currently in, and any levels above the current level on the table. However, the agent cannot access information on a level in the table that is lower than the level of the network in which the agent belongs.

The final factor that determines the type of network is the market condition. The market conditions determine the value of the end customer demand, which is imposed on the focal organization. There are five functions to describe the five different market conditions. The end customer demand is based on a total possible range of {0..2,000} to standardize the experiments. The demand functions are listed in the following table.

MARKET CONDITION	DEMAND FUNCTION	DEMAND RANGE
Stable	$a + e$	900..1,100
Increasing	$a[1 + (t / \# Cycles)] + e$	900..2,100
Decreasing	$a[1 - (t / \# Cycles)] + e$	1,100..0
Volatile	$a + E$	0..2,000
Seasonal	$if\ \frac{1}{3} \# Cycles < t < \frac{2}{3} \# Cycles, then = 2a$ *else = a*	1,000 or 2,000

Table 2-6 End customer demand by market.

In the table above:

- '*a*' is the initial production for each agent.
- '*t*' is the current cycle [1 .. 100,000].
- '*e*' is a randomly generated number [-100 .. 100]
- '*E*' is a randomly generated number [-1,000 .. 1,000]

The output of these demand functions is shown in the following charts.

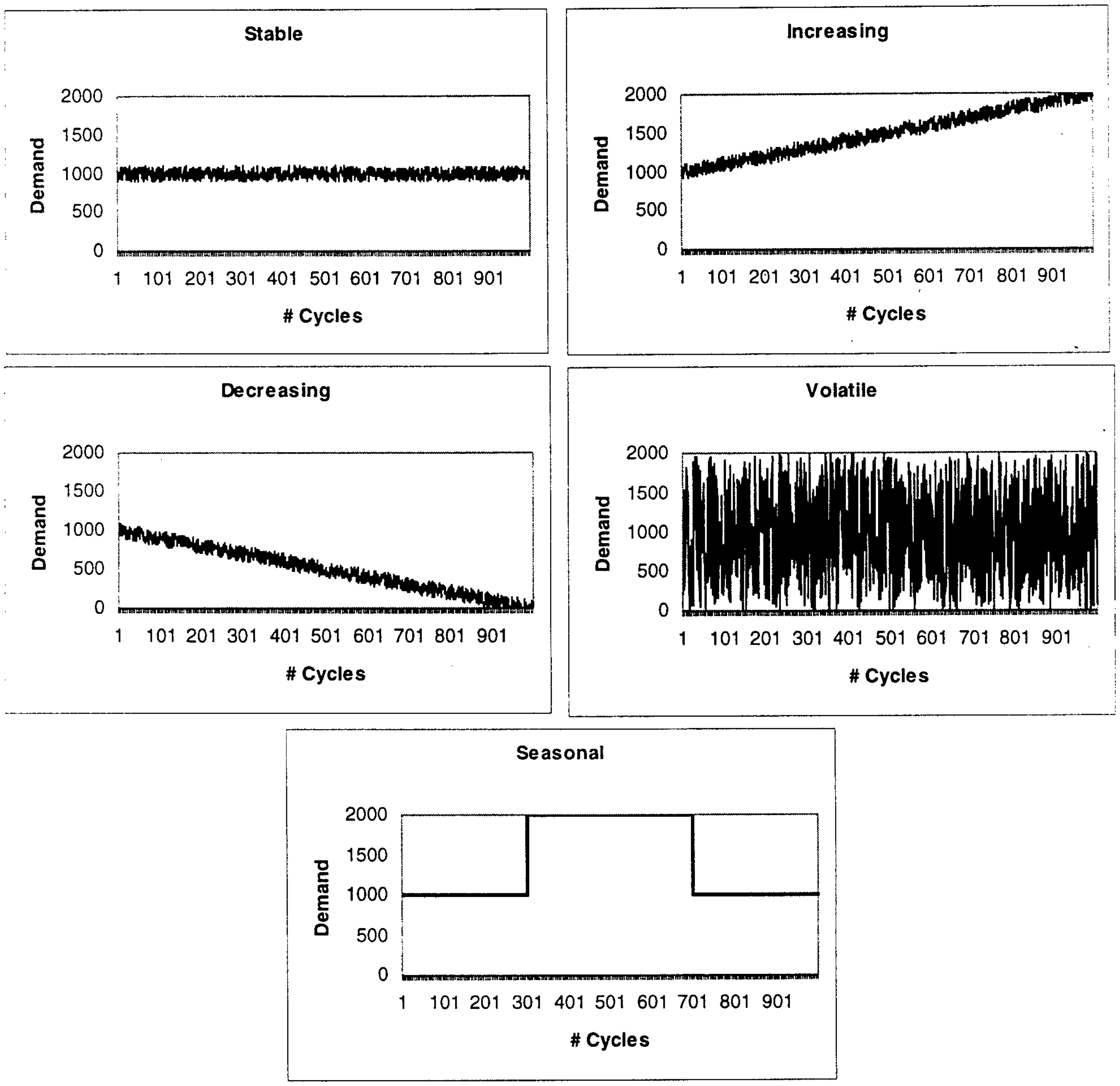


Figure 2-3 Output of customer demand functions.

2-2.3 Fitness Metric

The fitness metrics used for the network have constructs similar to those used for the individual agents, described in section 2-1.2. Similar to the individual agent's fitness functions, the following fitness functions consider zero optimal.

$$globalFitness = \left(\sum_{i=1}^{n} \left| endGap(i) \right| \right) / n$$

n = "number of agents in network"

$$networkFitness = \left(\sum_{i=1}^{n} localFitness(i) \right) / n$$

Here, the global fitness is a reflection of how well the agents in the network collectively meet the needs of the end customer. The network fitness is simply an average of the local fitness's of all agents in the network.

These two fitness metrics are the key to understanding the dynamics of the system as a whole. Although it would be beneficial to analyze each agent's local and regional fitness on an individual basis, in this project, we limit our analysis to the global and network fitness levels of the entire network and leave more in depth analysis to further study.

Chapter 3: Design of Simulation Experiments

We will run a series of simulation experiments that investigate the emergent behavior of a supply network enterprise under various conditions of three factors: inter-agent control method, supply network structure, change in internal dynamics and external environment. We will examine both the performance aspect and the structural aspect of the emergent behavior of the supply network enterprises. The inter-agent control methods will vary from only material flows, to material flows and information flows, and finally to material flows, information flows, and knowledge flows. For this one-year project effort, our network has a static physical structure. In future experiments, the supply network structures will differ in the number of tiers of suppliers for a focal organization, number of suppliers in each tier, the involvement (full or partial) of resources from a supplier in the supply network enterprise, and so on. Changes in internal dynamics and external environment will be set through the rate and type of changes within individual agents and from the external market environment.

In summary, in simulation experiments, we will investigate the following factors:

1) Inter-agent control methods for different degrees of agent autonomy such as sharing of state information among agents, conformance to a common fitness function by agents, and enforcement of the same internal model among agents; and
2) System structures determined by types and density of inter-connections among agents through material and information flows.

The design of these simulation experiments is based on a combination of current design techniques in supply networks, complex adaptive systems, multi-agent systems, production-distribution models, dynamic networks, self-organizing structures, networks exhibiting emergent behavior, distributed control systems, and discrete event systems. [1-10]

3-1 Initial Setup

In chapter 2, we defined state variables to describe the simulation system. These variables are summarized in the following table.

VARIABLE TYPE	VARIABLE	POSSIBLE VALUES / IMPLICATIONS
Independent (Network Level)	homogeneity	[homogeneous | semi-hetero | heterogeneous]
	level of autonomy	[low | high]
	information sharing	[none | regional | global]
	market condition	[stable | increasing | decreasing | volatile | seasonal]
Independent (Agent Level)	Role	[focal | 1Tier | 2Tier | ... | nTier]
	Internal Model	Defined by current set of building blocks.
	Building Blocks	Set of functions based on type of network.
	Connectivity	Number of connected suppliers.
Dependent (Network Level)	customer demand	Based on market conditions.
	global fitness	Average endGap of all agents in network.
	network fitness	Average local fitness of all agents in network.
Dependent (Agent Level)	production level	Based on homogeneity, autonomy, & info sharing.
	local fitness	Based on gap.
	regional fitness	Average local fitness of all agents in region.

Table 2-7 Summary of simulation state variables

Our simulation contains four independent variables. The number of possible values for these variables are: three, two, three, and five respectively. This gives us a total of 90 simulation states. Throughout the study, we narrow our focus to those states of greatest interest.

In those experiments where the agents' initial production value is the same (homogeneous and semi-hetero networks), we set the initial production to 1,000. Experiments are originally run for 1,000 cycles. We do some minor checking of differing cycle times as shown in the results in chapter 5.

3-2 Running the Experiments

Each of the simulation sessions processes a set of experiments based on the types of networks under investigation. Each experiment in the set is run for 1,000 cycles. Some experiments are run for 2,000 and 10,000 cycles to further investigate effects of simulation length. Each cycle of a simulation experiment consists of the following five (six) steps:

1. Calculate end customer demand
2. Process order through network
3. Record 3 dependent variable values per agent
4. Record global fitness and network fitness
5. Calculate next cycle's production level by agent
    a. If autonomy is low – focal sets production level across network
    b. If autonomy is high – each agent determines own production level
6. Calculate next cycle's control strategies using adaptive logic for each agent.

    This step is only used in Class 3 simulations (see below).

Following the completion of an experiment, the network and its agents are reset to their respective initial states before the next experiment in the set (or set of experiments) is run.

To single out specific research aims, we break the testing down into three experiment classes. For each class, changes are made to the experiment set up based on observed simulation results. Each class removes uninteresting aspects from the previous class, changes some aspects to refine results, and adds new aspects to further investigate interesting phenomenon. The classes are divided as follows:

Class 1 – Investigates the effects of market conditions, information sharing, autonomy, homogeneity (limited to the homogeneous and semi-hetero levels), and the costs associated with changing production levels.

Class 2 – Investigates the effects of noise present in customer demand functions, length of simulation (# of cycles), homogeneity (limited to the semi-hetero and hetero levels), and standardizing the change in production costs.

Class 3 – Investigates the effects of adding adaptive logic to allow agents the ability to adjust their control strategies.

These experiment classes are further defined in the following three sections.

3-2.1 Class 1 Experiments

In this class of experiments, all settings are as described previously. This class is used as a basis to determine which factors contribute greatest to the overall fitness of the system. During this class of experiments, we limit investigation into the homogeneity of the system to purely homogeneous and semi-hetero network types. We also concentrate our efforts on determining the magnitude of the market effect on the network performance. Furthermore, we investigate the effects of change in production costs factored into the fitness functions.

3-2.2 Class 2 Experiments

Because of class 1 analysis, further described in section 5-1, we make the following changes in class 2:

- Remove homogeneous network type
- Remove volatile market condition
- Remove change in production cost from local fitness function

We enhance the investigation of the homogeneity of the network by adding a heterogeneous network type to compare with the semi-heterogeneous network. We further investigate the effects of market conditions by removing the 'noise' from the demand by market formulas. This is accomplished by simply removing the random *e* and *E* from the first four market conditions. To balance the effect of removing costs of change in production levels, we alter the production functions to increase or decrease the production by a factor of ten each time the production requires a change.

Semi-hetero and heterogeneous networks: (X&Y are random) X={1..30} Y={1..6}

```
if |workingGap| > X for Y cycles
        newProduction = production +/- 10
```

This modification increases the number of cycles needed to catch up production, which "magnifies" the inner workings of the network. These new formulas make better sense for a network with no production costs since now production can only change by a pre-selected amount.

In addition to the above changes, we increase the length of a simulation by running the experiments for one, two, and ten thousand cycles. These results are described in section 5-2.

3-2.3 Class 3 Experiments

Because of class 2 analyses, further described in section 5-2, we keep all changes initiated in class 2. We also eliminate the experimental cycle length of 1,000 cycles for reasons given in section 5-3. Further changes in this class investigate the effect of adding adaptive control logic within individual agents. This control logic allows an agent to change its internal model by altering the control strategies within its current set of building blocks following each cycle depending on the following logic:

If newGap >= oldGap for 5 cycles; decrement X & Y

If newGap < oldGap for 5 cycles; increment X & Y

In these formulas, X and Y are from the production logic formulas given in table 2-1 for semi-hetero and heterogeneous network types. The gap is determined by the level of information sharing as described in table 2-5. This logic tells each agent when to trigger a control change. Each time an agent triggers a control change, the X and Y values are either incremented or decremented by one accordingly. Further investigation into heterogeneous complex adaptive networks would involve allowing varying adaptive logic across the network. This one-year project focuses on the effects of all agents sharing the same adaptive logic.

3-3 Collecting Data

Following each cycle, the dependant variables associated with the network and the individual agents that are updated include all fitness levels, agents' production levels, and agent control strategies when triggered. The latter only applies to class 3 experiments. The updating occurs in a sequential fashion such that any variables, which rely on other variables, are updated following the variables that they rely on as follows:

- As order is processed through network, for each agent:

- ✓ Set gap levels for all three gap functions
- ✓ Calculate local fitness for cycle

- Following order processing:
  - ✓ For all agents – set regional fitness
  - ✓ Set global fitness
  - ✓ Set network fitness
  - ✓ Set production level for all agents based on level of autonomy
  - ✓ If class three experiments, set control strategies if logic triggers a change for this cycle.

All data is collected in two formats for use in the two different types of data analysis described in chapter 4.

Chapter 4: Statistical Data Analysis

We investigate the non-linear time-series analysis techniques from chaos and dynamical systems theories and multivariate statistical analysis techniques, and transform them into the scalable, applicable techniques for analyzing the emergent behavior and temporal patterns of the supply network system using the data obtained from the simulation experiments. [7-14] In section 3-3, we discussed the data collection process for each simulation experiment. This collected data is represented in multiple formats that correspond to the desired method of analysis. The two main analysis goals for this project include the performance of the system, and detecting emergent behavior in the system.

We investigate how to represent the simulation data from the model of a supply network enterprise in a form that is acceptable to these analytical techniques, and how to reform and advance these analytical techniques so that they are scalable for real-time data analysis. Throughout the analysis process, techniques for analyzing data, as well as techniques for appropriately collecting data, are refined to enhance the validity and comprehension of each metric.

4-1 Measuring Performance

Performance is measured through the analysis of inherent patterns within the collected simulation data. To analyze the performance of the entire supply network enterprise, we focus our attention on the dependent variables: production, global fitness and network fitness. The data output from the simulation process is formatted in two ways to enable the import of data into graphing software to perform visual observations as well as the ability to import the collected data into statistic software for further analysis, including ANOVA analysis and Tukey HSD tests. The measure of performance allows the determination of control techniques and network types that enhance performance versus those that worsen it. Accurate analysis of the performance of the supply network enterprise serves to direct research into the appropriate areas of communication and control necessary to improve the performance of the system.

4-1.1 Visual Observations

For visual observations, the data is imported to a graphing software package, where it is plotted for analysis. For each experiment, we create production and fitness plots. The production charts include the production level of each agent in the network over the cycles in the simulation experiment. With these plots, we can visualize the effect of information sharing, homogeneity, autonomy, and control functions on the production of the agent system. Global fitness and network fitness plotted together for each experiment allow observation of the comparison and interaction of these two metrics. Over a set of experiments, we create separate plots for global and network fitness. Each chart shows how the respective fitness level varies under differing network types and simulation classes.

Visual observations provide a dual purpose. They allow the confirmation of known or suspected system behavior, and therefore give assurance that the simulation software is behaving as expected. Furthermore, visual observations are an easy way to detect areas of study that are deserving of further attention, as well as those that do not appear to contain any further enlightening discoveries. Visual observation is used as a first step in our data analysis for the benefit of both of these principles.

4-1.2 ANOVA Analysis

An analysis of variance (ANOVA) is used to test for statistically significant differences in means. The testing is based on the partitioning of the variances. [10] The results of analysis show us the magnitude (F) of the relationships of variables, or differences of means, and the statistical significance (p-value) of each result. The p-value of a result is a measure of how significant the observed relationship or difference really is. This value is the probability that the observation occurred by pure chance. In our study, we only consider observed results significant if the corresponding p-value is < .05. ANOVA allows us to determine which of the four factors given in table 2-2 contribute greatest to the observed effect on the different fitness levels.

Because the supply network enterprise is multivariate, we use multi-factor ANOVA to produce tabular and graphical comparisons of the effects of the multiple factors on a single dependent variable. We use graphs produced from ANOVA results to show the 4-way interaction between factors for both the global and network fitness metrics in chapter 5. Other results show us the effects of 2-way and 3-way interactions for more detailed observations of the relationships between factors.

For each of our experiments, the results include the level of each fitness metric for every cycle in the experiment. For example, after 1,000 cycles, we have 1,000 local fitness points for each agent in the network, 1,000 global fitness points, and 1,000 network fitness points. We process the ANOVA analysis for the local fitness of every agent, and the global and network fitness metrics. For this phase of the project, we focus on the global and network fitness levels. For each of these fitness metrics, we produce a summary of all effects that shows 1-way, 2-way, 3-way, and 4-way interactions between the four factors for that fitness metric. This summary includes the F and p-levels as described above. Next, we create a graph of the 4-way interaction to visualize the interaction between factors. Finally, we process Tukey HSD tests (described below) for all statistically significant results in the summary table to further analyze the effects of the factors and their interaction with each other.

Results for each experiment are compared with other experiment results within the same class to identify important factors and interesting behavior. The results of a class of experiments are then compared to another class for identifying behavioral changes that occurred by instituting changes to the simulation. This process allows us to refine our simulation and analysis techniques.

4-1.3 Tukey HSD Testing

A post-hoc comparison of means is given by the Tukey honest significant difference (HSD) test. This test allows the grouping of means to see which groups are particularly different from each other. The use of post-hoc comparison techniques is preferred over other techniques, such as the t-test for independent samples, because they take into account that more than two samples were taken. This prevents the possibility of reported probability levels overestimating the statistical significance of the mean differences. With the Tukey tests, we can see how the factors work together in producing the statistically significant effects observed in the ANOVA testing process. Many of the observations and discussions given in chapter 5 result from analysis of the Tukey HSD tests results for each

The format of the Tukey HSD test allows processing tests with 1-way, 2-way, 3-way, and 4-way interactions. This way, we can run a Tukey test on any of the observed statistically significant interactions found in the ANOVA summary described above. The output is a table with rows indicating all possible combinations of factors. These rows are ordered such that the combination resulting in the smallest mean is first, and the combination resulting in the largest mean is last. Through this format, we can look at a Tukey result and see immediately which combination of factors gives the best and worst performance values for the network. Furthermore, the Tukey HSD results break up the rows into groups, which have means that are statistically similar when compared to the other combinations of factors. With this, it is easy to identify which factors contribute the most to a major change in performance level, and which factors only contribute to minor performance changes.

4-2 Detecting Emergent Behavior

We consider a number of non-linear time-series analysis techniques such as the delay-coordinate embedding technique to reconstruct the phase space from time-series data, estimate the dimensionality of the system state via the correlations dimension, and thus detect emergent states of the system, as well as multivariate statistical analysis techniques such as clustering and classification techniques. [9-14] The importance of detecting emergent behavior is abundant. In one case, if the emergent behavior of the network is producing damaging effects to the overall performance of the system, control techniques can be applied to drive the network out of the emergent state. In another case, if a network is exhibiting emergent behavior that is proving beneficial to system performance, alternate control strategies (or perhaps the absence of control), can be applied to keep the system in the emergent state. Furthermore, in a proactive way, if we can find the combinations of factors that drive a network into an emergent state, we can either prevent or encourage this combination of factors depending on whether the emergent state is considered good or bad for the network.

4-2.1 Visual Observations

As with the performance measurement techniques, we begin by making visual observations in an attempt to detect emergent behavior in the system. These observations assure us that our simulation is behaving as expected and that the data collected from the system is adequate to subject to more complex emergent behavior analysis techniques. To detect the presence of emergent behavior, we plot the production levels of each agent over the course of a simulation and compare the series. If the agents' production levels appear sporadic and undeterminable, we predict that the simulation will not exhibit emergent behavior. However, if the agents' production levels seem to converge at any point, we guess that the simulation will not exhibit emergent behavior. With the knowledge gained from these visual observations, we can continue our study into the emergent behavior of the network through more sophisticated techniques.

We observe the output of each experiment with respect to global and network fitness. In the cases where the output appears chaotic, we determine that chaotic time series analysis is appropriate.

4-2.2 Chaotic Time Series Analysis

Chaotic time series analysis is used to detect emergent behavior in a system by identifying the deterministic origin of a time series with chaotic underlying dynamics. By estimating the dimensionality of the stochastic process, it is possible to detect a chaotic time series in the output of the process. In chaotic time series analysis, delay coordinates are commonly used to reconstruct an image of a dynamic system. [11] The correlation dimension (D_2) of the image of an attractor in the original dynamic system can be estimated by a technique presented in [11-14] which uses the embedding dimension and delay time coordinates to estimate D_2. This technique applies the selection of appropriate values for the embedding dimension and delay time to the Grassberger-Procaccia algorithm.

This algorithm evaluates D_2 using the probability that a randomly chosen pair of points will be separated by a distance less than ε on the attractor [11-14]

$$C_N(\varepsilon) = \frac{2}{N(N-1)} \sum_{j=1}^{N} \sum_{i=j+1}^{N} \Theta(\varepsilon - \|x_i - x_j\|)$$

where Θ is the Heaviside function [$\Theta(x)=1$ if $x \geq 0$, and $\Theta(x)=0$ otherwise]

and $\|x\| = max\{\|x_i\| : 1 \leq i \leq m\}$, where m is the embedding dimension

Then the correlation dimension is given by:

$$D_2 = \lim_{\varepsilon \to 0} \lim_{N \to \infty} \frac{\log C_N(\varepsilon)}{\log(\varepsilon)}$$

For our application, we designate the correlation sum as $C_N(\varepsilon,\tau,m)$. [11] Where τ is a constant time-delay selected by observing a graph of the stochastic process output, and *m* is a variable representing the embedding dimension. We plot the slope of the linear portion of D_2 over increasing values of *m* to detect emergent behavior within the supply network enterprise. If emergent behavior exists in the network, then for some dimension *m*, the slope of D_2 is the same for all $x \geq m$. Typically, the slope of D_2 increases with *m* until this plateau is reached. [11]

In detecting emergent behavior, we run this algorithm on the global fitness and network fitness outputs of each simulation experiment that exhibits chaotic time series output as observed in the visual observations of 4-2.1.

Chapter 5: Results and Discussion

The results presented here are separated by experiment class. The classes of experiments are described in section 3-2. These classes allow us to single out specific areas of interest for differing research aims. Within each sub-section of this chapter, we describe results relating to key areas of investigation for that class, changes made to the simulation model from previous classes, and results that may warrant changes in future classes.

5-1 Class 1 Experiments

Our initial set of experiments show that the effect of a volatile market is highly significant. The volatile market shows significantly worse performance levels than the other four markets (stable, increasing, decreasing, and seasonal) for both global and network fitness. For the global fitness, market level is the only significant factor. We determine that fitness is driven by the market due to a cancellation effect (when one tier has poor fitness, another has good fitness).

For network fitness, all four factors are significant. The significant factors, when analyzed unaccompanied (1-way interaction), which result in the best performance are homogeneous; low autonomy; and global information sharing. We observe that a homogeneous, high autonomy network with global information sharing can achieve the same effects as a homogeneous, low autonomy network. Furthermore, a semi-hetero, high autonomy network achieves best fitness when each agent reacts only to its immediate customer (no information sharing). Another significant observation is that increasing the level of information sharing among autonomous agents improves network fitness in homogeneous networks, and worsens it in semi-hetero networks.

In the graphs that follow, the first four pairs of graphs represent global fitness, and the next four pairs represent network fitness. The pairs represent autonomy (low and high), and there is one pair for each of four market conditions (stable, increasing, decreasing, and volatile). The y-axis of each graph is fitness level, and the x-axis is the information sharing level (none, regional, global). The lines represent the homogeneity of the network (homogeneous and semi-hetero).

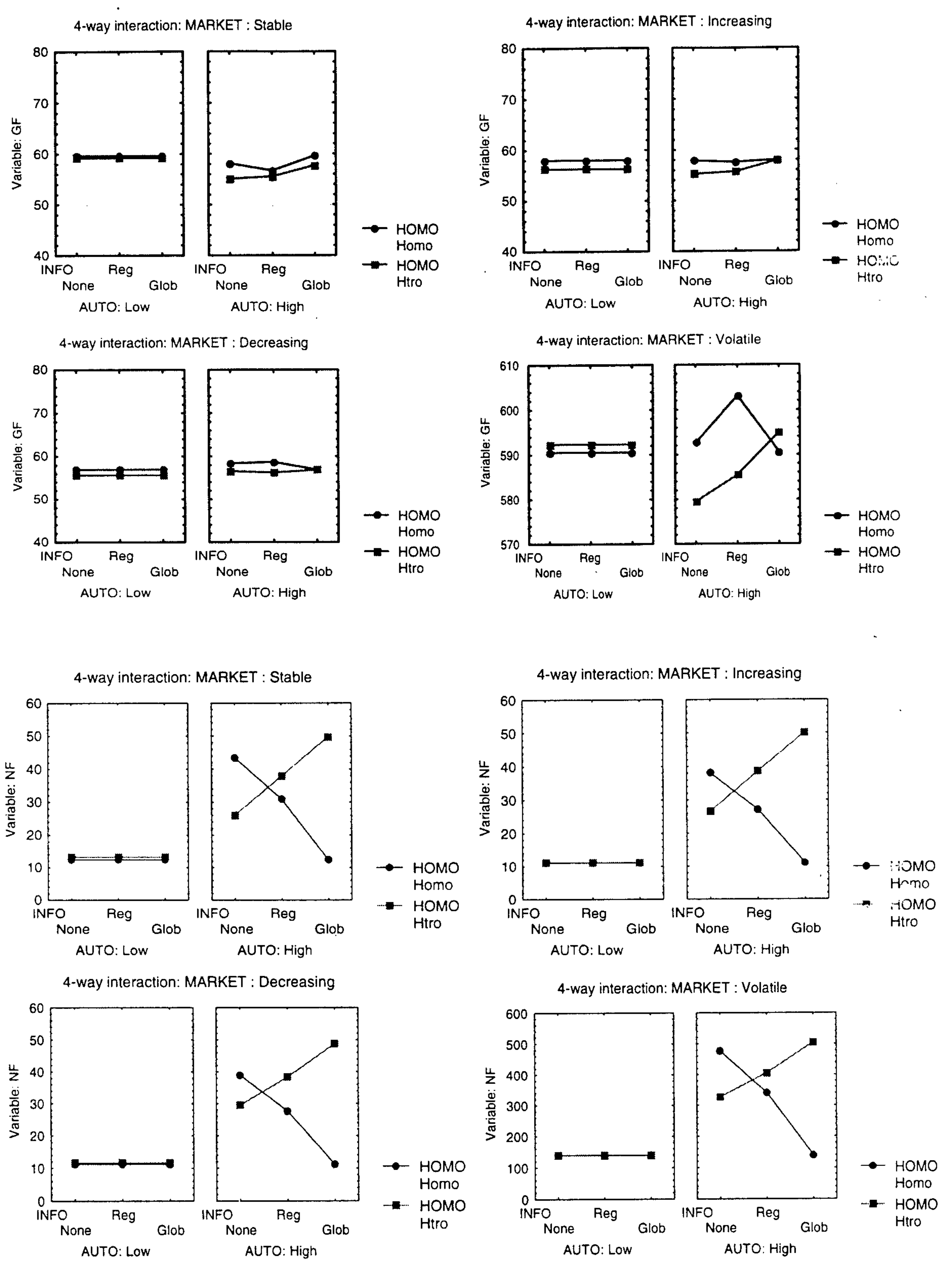


Figure 2-4 Global and network fitness in class 1 experiments.

Observe that for each market condition, the underlying patterns of variable interaction are similar, however, the values of the fitness levels are significantly higher (worse) in the volatile market. The graphs also clearly identify the cross-relationship between homogeneity and information sharing. Chaotic time-series analysis shows that this network is a purely stochastic system exhibiting no emergent behavior.

In the seasonal market, for global fitness all factors except homogeneity are significant. The best significant 1-way factors are seasonal and low autonomy. In the network fitness, all factors are significant. The best significant 1-way factors are seasonal, homogeneous and low autonomy. Information sharing improves both fitness measures in the seasonal market for homogeneous and semi-hetero autonomous networks.

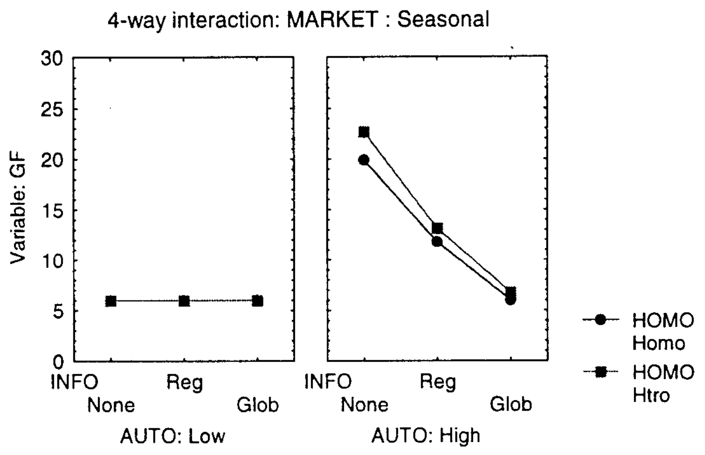


Figure 2-5 Global fitness is a seasonal market.

For all simulations, low autonomy provides better performance overall. The seasonal market with its absence of noise provides the best fitness levels. Observe also that the cross-interaction between homogeneity and information sharing does not exist without the noise present in other market conditions.

Recalling that our fitness levels take into account the agent's ability to meet customer demands, as well as the cost of changing fitness levels, we investigate the effect these costs have on the overall fitness.

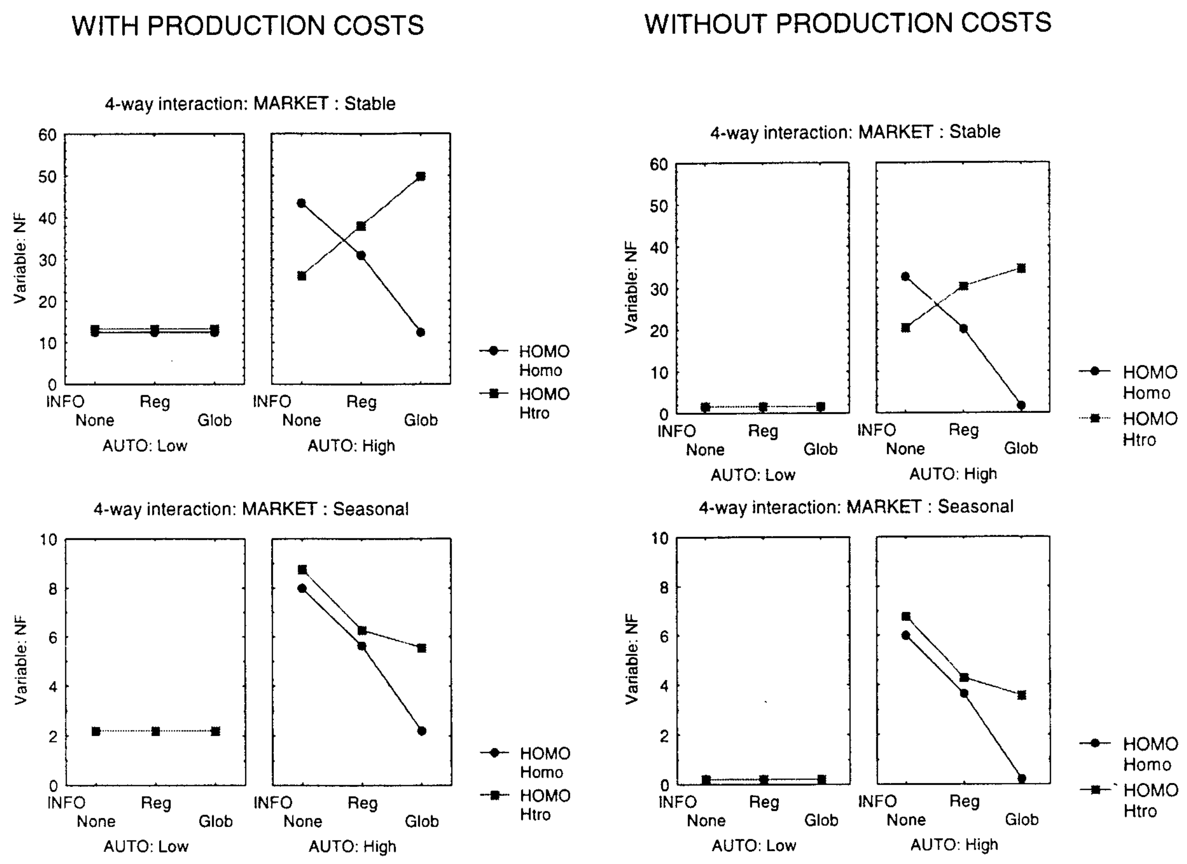


Figure 2-6 Effects of production costs on the fitness levels.

As shown in this chart, the effect of calculating the change in production costs into the fitness metric does not have a significant effect on the overall performance of the network.

5-2 Class 2 Experiments

The experiments in this class follow the changes to the model outlined in 3-2.2. The results of these modifications are as follows:

- Global Fitness
  - Increased information sharing improves fitness in an autonomous network.
  - Best significant 1-way: low autonomy; global info sharing
- Network Fitness

- Fitness worsens or stays the same with increased information sharing in an autonomous network.
- Best significant 1-way: low autonomy; no info sharing

- Both fitness measures
  - All factors are significant.
  - Seasonal market shows the worst performance levels.

In the graphs that follow, the first set of four graphs is global fitness and the second set is network fitness. In this example, we only look at the purely heterogeneous network to compare with semi-hetero graphs from the previous class of experiments. Therefore, the lines on the graphs now represent low and high autonomy so that we can measure the significance of this factor. The remaining parts of the graphs remain the same as the last set.

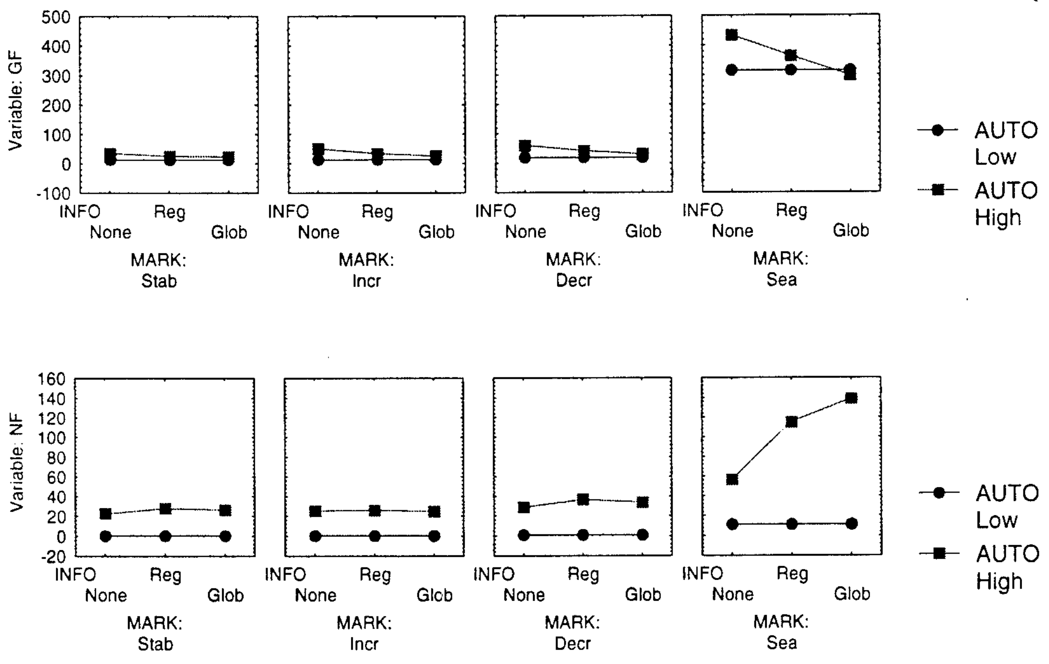


Figure 2-7 Global and network fitness in class 2 experiments.

From these graphs, we can see the significant effect of the changes outlined in section 3-2.2. In further experiments, we ran the simulation for 2,000 and 10,000 cycles. As the number of cycles increased, we find that the overall fitness levels improve (with very small changes between 1,000 and 2,000 and larger changes between 2,000 and 10,000). However, the overall Interaction between the factors remains the same. Additionally, the negative effect of information sharing on network fitness is decreased (positive in some markets) at 10,000 cycles.

With the removal of noise from the demand functions, we find that the output of our simulation experiments no longer has an underlying stochastic structure, which nullifies the need for detecting emergent behavior. We continue our in depth study into the workings of the supply network enterprise to complete this one-year effort with the aim to refine our model before reverting back to a more complex customer demand market. We save the further research into the analysis and control of emergent behavior to future study, and continue investigating our model.

5-3 Class 3 Experiments

In order to witness the effects of adding adaptive logic to the agents, we run the simulations for 2,000 cycles. As observed in class 2 experiments, this will not degrade our result comparisons with the other classes. The charts showing the global and network fitness of this class are set up in the same way as the class 2 experiment charts. As with the class 2 experiments, all factors are significant for both global and network fitness measures.

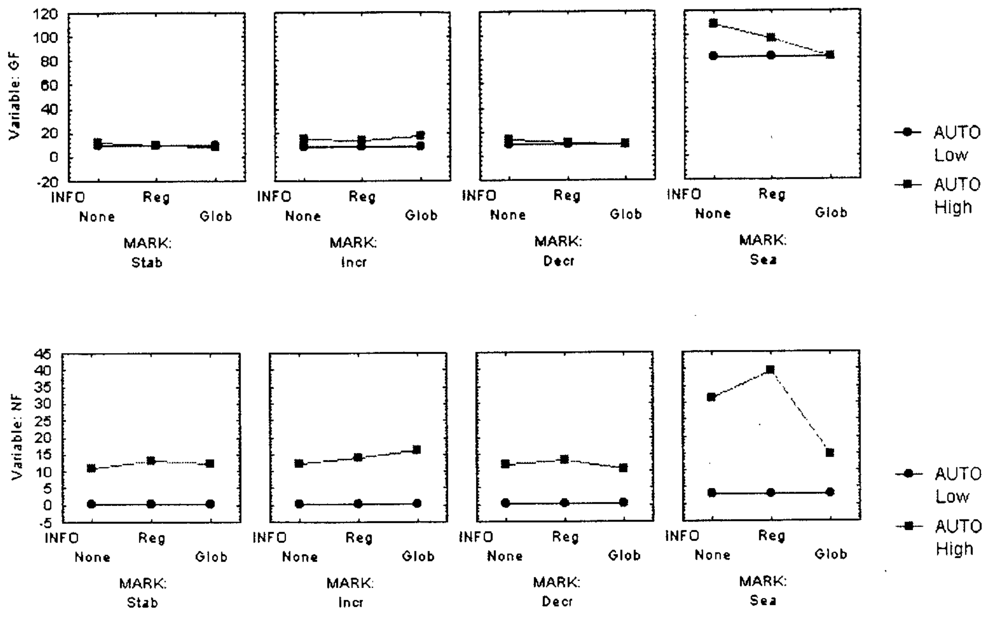


Figure 2-8 Global and network fitness in class 3 experiments.

We observe from these charts, that the overall pattern of variable interaction remains the same as in class 2 experiments, with the exception of the network fitness in a seasonal market. However, the level of fitness is significantly lower (better) in all cases. This follows from the scale along the y-axis. The interesting observation here is that the changes made in class three simulations have a significant positive impact on the network fitness level in a seasonal market with high autonomy, and global information sharing.

Chapter 6: Conclusions

In this project, we have shown a method for designing a model of a supply network enterprise. With this model, we use our simulation method to analyze the behavior of the supply network enterprise. Our analysis techniques prove creditable as shown in the simulation results of chapter 5. This one-year project has culminated in the design and implementation of a simulation model, the refinement of the model through simulation processing, the application of analysis techniques to understand the simulation results, and the resulting observations of a complex, dynamic supply network enterprise. To summarize our results, we discovered the following:

- ✓ Volatile markets have the most negative impact on the system performance.
- ✓ Low autonomy has the most positive impact overall on the system performance.
- ✓ A homogeneous, high autonomy network with global information sharing can achieve the same performance as a homogeneous, low autonomy network.
- ✓ Noise present in customer demand creates a cross-relationship between homogeneity and the level of information sharing.
- ✓ Performance does not always improve with increased information sharing in an autonomous network.
- ✓ With adaptive logic, agents can improve their fitness levels, thereby improving the performance of the entire system.

These observations provide a basis for observing the effects of our simulation model in chaotic environments.

References

1. Choi T, Dooley K, Rungtusanatham M. *Supply Networks and Complex Adaptive Systems: Control versus Emergence.* Journal of Operations Management 19: 351-366, Elsevier Science B.V., The Netherlands, 2001.
2. Swaminathan J, Smith S, Sadeh N. *Modeling Supply Chain Dynamics: A Multiagent Approach.* Decision Sciences, Volume 29 Number 3, Summer 1998.
3. Chang M, Harrigton, Jr. J. *Centralization vs. Decentralization in a Multi-Unit Organization: A Computational Model of a Retail Chain as a Multi-Agent Adaptive System,* Cleveland State University, September 1999.
4. Larsen E, Morecroft J, Thomsen. *Complex Behaviour in a production distribution model,* European Journal of Operational Research 119: 61-74, Elsevier Science B.V., The Netherlands, 1999.
5. Lin F, Huang S, Lin S. *The Effects of Information Sharing on Supply Chain Performance in Electronic Commerce,* National Sun Yat-sen University, Kaohsiung, Taiwan 804 R.O.C.
6. Periorellis P, Dobson J, *Modelling Dynamic Networks Advantages and Limitations of the O-O Approach,* Center for Software Reliability, University of Newcastle, U.K.
7. Banks J, Carson II, J. *Discrete-Event System Simulation,* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1984.
8. Walker C, Dooley K. *The Stability of Self-Organized Rule-Following Work Teams,* Kluwer Academic Publishers, The Netherlands, 1999.

9. Kogut B. *The Network as Knowledge: Generative Rules and the Emergence of Structure,* Strategic Management Journal, 21:405-425, 2000.

10. Parunak H, VanderBok R. *Managing Emergent Behavior in Distributed Control Systems,* Industrial Technology Institute, Ann Arbor, MI.

11. Lai Y, Lerner D. *Effective scaling regime for computing the correlation dimension from chaotic time series,* Physica D 115: 1-18, Elsevier Science B.V., The Netherlands, 1998.

12. Lai Y, Osorio I, Harrison M, Frei M. *Correlation-dimension and autocorrelation fluctuations in seizure dynamics,* Arizona State University, Tempe, AZ, 2001.

13. Lai Y, Kostelich E. *Are spatiotemporal dynamics detectable from delay-coordinate embedding of scalar time series?* Arizona State University, Tempe, AZ, 2001.

14. Dhamala M, Lai Y, Kostelich E. *Analyses of transient chaotic time series,* Georgia Institute of Technology, Atlanta, GA, 2001.